

6 Todos somos iguales

Revista de Software Libre ATIX 2008

Equipo Editorial

Colaboradores

Dirección y Coordinación General

Esteban Saavedra López (jesaavedra@opentelematics.org)

Diseño y Maquetación

Jenny Saavedra López (jennysaavedra@hotmail.com)

Esteban Saavedra López (jesaavedra@opentelematics.org)

Herramientas

La edición de esta revista fue realizada de forma integra haciendo uso de Software Libre









Palabra quechua,
con un sentimiento profundo
y con gran significado filosófico

El que lo sabe

El que lo intenta

El que lo puede

El que lo logra

El que lo sabe, el que lo intenta, el que lo puede, el que lo logra

Editorial

Desde el principio de los tiempos el hombre siempre buscó la igualdad en todo, en derechos y en obligaciones; lastimosamente en muchos lugares y en varios aspectos después de tantos años, aún no se han podido sopesar problemas de desigualdad, pero estamos seguros que más temprano que tarde éstos tendrán un final feliz.

Hoy en día dentro de una vida llena de actividades sociales, económicas y académicas, todos luchan por conseguir esa preciada igualdad; muchos estamos convencidos que una forma de establecer la misma, es la de compartir conocimiento, ya que el compartir representa articular la enseñanza y el aprendizaje con un objetivo en particular "El bienestar común", conocedores que el conocimiento ha llevado, lleva y llevará a construir un futuro mejor para todos.

Todos somos iguales, un título que refleja de forma clara y concisa lo que todo hombre , mujer, niño o anciano desea, saber que todos somos iguales; a lo largo de este tiempo dentro de la **Revista Atix**, nos hemos dado cuenta que el dar oportunidad a todos (experimentados y novatos) nos ha permitido contribuir a fortalecer ese espíritu de igualdad, que tanta falta nos hace a todos los seres en este mundo. Por todo ésto y por mucho más, instamos a todos a construir entornos de igualdad , sin discriminar ningún criterio en particular.

En éste quinto número ponderamos la continuidad de varios de nuestros artículos por parte de sus respectivos autores, por el interés que han cobrado estos en nuestros lectores quienes fielmente continúan número a número todos nuestros artículos.

El ser todos iguales, nos llevará a tener y vivir un futuro mejor.

Bienvenidos a nuestro sexto número

Esteban Saavedra López
Director y Coordinador General

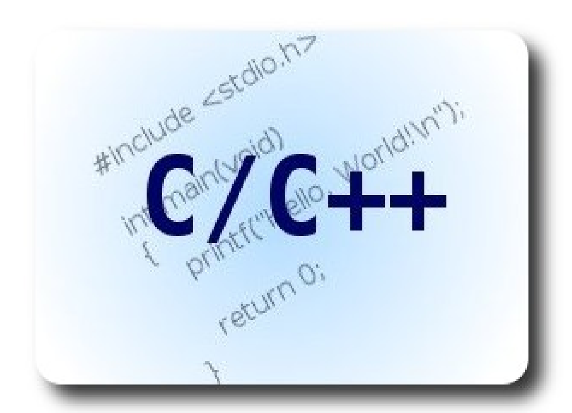
Contenido

Liberado el 17 de diciembre de 2008



Utilizando herramientas de desarrollo C++ en Linux (2da parte)

Desarrollar y construir sistemas en entornos *NIX puede llegar a parecer algo dificultoso al inicio, sin embargo, con el uso adecuado de las herramientas que proveen los entornos GNU, ésta se vuelve una tarea sencilla.



Introducción

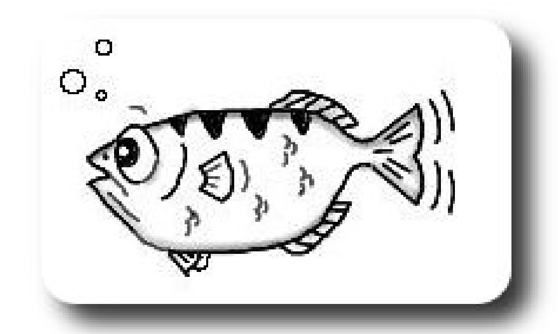
En el anterior número vimos la forma de compilar manualmente un pequeño ejecutable de distintas maneras, además de observar algunos estándares de codificación utilizados en la mayoría de proyectos escritos en C/C++, tal como la separación entre declaración y definición.

En esta ocasión estudiaremos algunas herramientas de apoyo a la codificación:

El depurador estándar de GNU: GNU Debugger y la herramienta de automatización orientada a verificar dependencias: Make

Depuración de programas: gdb

El depurador de GNU (GNU Debugger : gdb) es una herramienta estándar de alta versatilidad escrita por el mismísimo Richard Stallman para el sistema GNU/Linux. gdb es una herramienta para línea de comandos, por tanto no posee un front-end gráfico, sin embargo, existen aplicaciones como DDD (Data Display Debugger), que permite hacer una depuración gráfica, aprovechando todos los comandos de gdb.



Archer, la mascota del proyecto gdb

Este tipo de pez es famoso por cazar insectos (bugs) desde el agua.



Logo del proyecto DDD(Data Display Debugger)

Cargar programa en memoria

Para probar este depurador vamos a utilizar el siguiente programa:

programa.cpp

```
 1  int otra_funcion(int w, int u){
 2          int ret = w;
 3          ret = ret * u;
 4          return ret;
 5  }
 6
 7  int funcion(int x){
 8          int suma=3;
 9          for(int j=0;j<3;j++)
10                  suma += otra_funcion(suma,x);
11
12          return suma;
13 }
14
15 int main(void){
16          int a,b,c;
17          for(int i=0;i<10;i++){
18                  a = i+1;
19                  b = a+i;
20                  c = funcion(a*b);
21          }
22
23          return 0;
24 }
```

Por si solo, este programa no muestra absolutamente nada, sin embargo, los cambios de variables, los ciclos y las llamadas a funciones es lo que nos interesa. Primero debemos crear el ejecutable, además debemos especificar al compilador que este debe ser “depurable”. Esto se consigue añadiendo la opción -g , que incorpora información extra a los archivos generados, la cual establece un vínculo entre el archivo binario producido, los archivos fuente y sus respectivas líneas de código.

```
arn@localhost:~$ g++ programa.cpp -o programa -g
```

Es el momento de preparar y cargar el ejecutable en memoria del depurador de la siguiente manera:

```
arn@localhost:~$ gdb ./programa
GNU gdb 6.8-debian...
....
(gdb)
```

La nueva línea de comandos se cambiará a (gdb) que es donde se escriben las órdenes de depuración. Si ejecutamos el programa en este punto, la ejecución llegará al final sin que saquemos beneficio del depurador, debemos primero establecer breakpoints o lugares donde el programa se detendrá para inspeccionarlo.

Un buen punto para poner un breakpoint es en el inicio mismo del programa, de esta forma:

```
(gdb) break main
Breakpoint 1 at 0x804844b: file programa.cpp, line 17.
```

La primera línea de código en ejecutarse será la número 17, aunque en la línea 16 exista código, ésta únicamente declara variables, no realiza una acción como tal. Después de establecer el breakpoint, se ejecuta el programa con la orden run:

```
(gdb) run
Starting program: /home/arn/programa
Breakpoint 1, main () at programa.cpp:17
17 for(int i=0;i<10;i++){
```

gdb muestra el contenido de la línea de código que está a punto de ejecutarse.

Para hacerlo y saltar a la siguiente utilizamos la orden next:

```
(gdb) next
18 a = i+1;
```

Inspección de variables

En este punto vamos a inspeccionar la variable que controla el ciclo:

utilizando la orden print

```
(gdb) (gdb)
$1 = 0
```

El valor de i en estos momentos es cero. Ejecutemos las siguientes tres líneas de código. En vez de next, pulsaremos simplemente la letra n, una comodidad que nos proporciona gdb para los comandos en su forma simplificada:

```
(gdb) n
19 b = a+i;
(gdb) n
20 c = funcion(a*b);
(gdb) n
17 for(int i=0;i<10;i++){
```

En este punto las variables a, b y c ya cambiaron, así que podemos inspeccionarlas, en vez de print, usaremos su equivalente corto p:

```
(gdb) p a
$2 = 1
(gdb) p b
$3 = 1
(gdb) p c
$4 = 24
```

Lo anterior nos indica que la variable a tiene el valor de 1, b vale 1 y c vale 24.

Una característica poderosa de gdb es la capacidad de evaluar expresiones arbitrarias.

```
(gdb) p a * 2 + 4 - b
$5 = 5
```

La evaluación de la anterior expresión da como resultado 5.

También podemos cambiar los valores según nuestras necesidades, alterando el flujo normal de la ejecución del programa para realizar pruebas. Por ejemplo, cambiemos el valor de i con el comando set variable:

```
(gdb) p i
$6 = 0
(gdb) set variable i = 6
(gdb) p i
$7 = 6
```

Antes de cambiar el valor de la variable i , su valor era cero. Después de hacerlo comprobamos que ahora vale 6.

La inspección y manipulación de variables puede ser realizada en cualquier punto de la depuración con las mismas órdenes ya mencionadas.

Uso de frames

Hasta este momento, hemos inspeccionado las variables en un solo contexto (main). Las variables de este contexto son i , a, b y c, en la línea 20 se utiliza una llamada a una función, sin embargo, utilizando la orden next, simplemente ejecutamos dicha línea de código y saltamos hacia la siguiente sin entrar en detalles de lo que contiene la función, tampoco podemos depurar lo que hace.

Para hacerlo vamos a utilizar la orden next hasta llegar a dicha línea 20

```
(gdb) n
18 a = i+1;
(gdb) n
19 b = a+i;
(gdb) n
20 c = funcion(a*b);
```

Antes de ingresar a depurar la función, comprobemos el valor del parámetro a*b que se le está pasando:

```
(gdb) p a*b
$8 = 120
```

También comprobaremos el contexto donde nos encontramos actualmente (main), utilizando la orden frame, el cual nos da la información de la línea de código que está a punto de ejecutarse, y el contexto :

```
(gdb) frame
#0 main () at programa.cpp:20
20 c = funcion(a*b);
```

También podemos ver todas las variables locales del contexto main utilizando la orden info locals:

```
(gdb) info locals
i = 7
a = 8
b = 15
c = 24
```

Es el momento de utilizar la orden step para

hacer un salto hacia dentro de función:

```
(gdb) step
funcion (x=120) at programa.cpp:8
8 int suma=3;
```

Adelantemos unas cuantas líneas de código (4) con la orden next, y noten que esta vez la depuración es dentro de la función.

```
(gdb) n
9 for(int j=0;j<3;j++)
(gdb) n
10 suma += otra_funcion(suma,x);
(gdb) n
9 for(int j=0;j<3;j++)
(gdb) n
10 suma += otra_funcion(suma,x);
```

Es un buen momento para inspeccionar las variables locales utilizando info locals:

```
(gdb) info locals
j = 1
suma = 363
```

y también para verificar el contexto actual con frame:

```
(gdb) frame
#0 funcion (x=120) at programa.cpp:10
10 suma += otra_funcion(suma,x);
```

Entremos un nivel más adentro en las llamadas a funciones, esta vez dentro de otra función, con la orden step

```
(gdb) step
otra_funcion (w=363, u=120) at
programa.cpp:2
2 int ret = w;
```

Hagamos una pequeña pausa aquí para explicar algunos conceptos importantes:

En este instante, estamos en el tercer nivel de llamadas a funciones, es decir, la función principal **`main`** hizo la llamada a la función denominada **`funcion`**, y a su vez, esta hizo la llamada a la función **`otra_funcion`**. Cada vez que llamamos a una función, el estado del depurador cambia de contexto. Por ejemplo, en el contexto de **`main`** existen las variables a, b y c, pero dentro de **`funcion`** dichas variables no existen, pero sí la variable suma, y a su vez, dentro del contexto de otra función, existen las variables u y w, y las demás son invisibles para este nuevo contexto.

Cada uno de estos contextos son el realidad lo que se denomina stack frames, o contextos de pila, cada uno con su propia información. La pila de llamadas a funciones es el stack, que guarda el orden de invocaciones. Muchas veces es útil investigar dicha pila para saber todos los contextos , puesto que en un programa de mayores dimensiones, la pila de llamadas puede ser bastante grande, y es importante saber la información que guarda.

Volviendo a la parte práctica, obtendremos la información de la pila y de todos los stack frames con la orden backtrace:

```
(gdb) backtrace
#0 otra_funcion (w=363, u=120) at
programa.cpp:2
#1 0x08048428 in funcion (x=120) at
programa.cpp:10
#2 0x08048475 in main () at
programa.cpp:20
```

Es posible cambiar el frame actual para inspeccionar valores de otro frame, por ejemplo, inspeccionar el valor de la variable a al momento de la última llamada de **`funcion`** en **`main`**. Si intentamos acceder directamente, pasar lo siguiente:

```
(gdb) p a
No symbol "a" in current context.
```

La variable a no existe en el contexto actual, así que debemos saltar al contexto de **`main`** (que está numerado como #2). Usaremos la orden frame para cambiar de contexto:

```
(gdb) frame 2
#2 0x08048475 in main () at
programa.cpp:20
20 c = funcion(a*b);
```

Aquí podemos inspeccionar la variable:

```
(gdb) p a
$9 = 8
```

De este modo podemos movernos por todos los stack frames para averiguar su información en cualquier instante.

Breakpoints

Los breakpoints son elementos de depuración que permiten detener el programa en una línea de código predeterminada. Por ejemplo, existe un breakpoint al inicio de la función **main** (la que pusimos al inicio). Pongamos un breakpoint en la línea 19:

```
(gdb) break 19
Breakpoint 2 at 0x804845d: file
programa.cpp, line 19.
```

A continuación hagamos que el programa siga corriendo , hasta llegar al primer breakpoint que encuentre usando la orden continue:

```
(gdb) continue
Continuing.
Breakpoint 2, main () at programa.cpp:19
19 b = a+i;
```

No importa el contexto en que se encuentren los breakpoints, el programa se detendrá de todas formas, para comprobarlo pongamos un breakpoint dentro de **otra_funcion**:

```
(gdb) break otra_funcion
Breakpoint 3 at 0x80483ea: file
programa.cpp, line 2.
```

Y dejamos que el programa siga su curso con continue:

```
(gdb) continue
Continuing.
Breakpoint 3, otra_funcion (w=3, u=153)
at programa.cpp:2
2 int ret = w;
```

Para obtener una lista de todos los breakpoints activos, usemos la orden info breakpoints

```
(gdb) info breakpoints
Num          Type              Disp Enb
Address        What
1            breakpoint        keep y
0x0804844b in main at programa.cpp:17
breakpoint already hit       1 time
2            breakpoint        keep y
0x0804845d in main at programa.cpp:19
breakpoint already hit       1 time
3            breakpoint        keep y
0x080483ea in otra_funcion(int, int) at
programa.cpp:2
breakpoint already hit       1 time
```

Hemos visto una serie de comandos básicos que posee gdb, tal vez solo raspando la superficie de los poderes increíbles que nos puede proporcionar este fantástico depurador. Una lista más completa puede ser obtenida con la orden help, o especificando la clase de ayuda, por ejemplo help breakpoints.

GNU Make

Make es una herramienta que controla la compilación de ejecutables y otros archivos similares desde sus ficheros fuente. Make se encarga automáticamente de comprobar que archivos son necesarios compilar y cómo hacerlo.

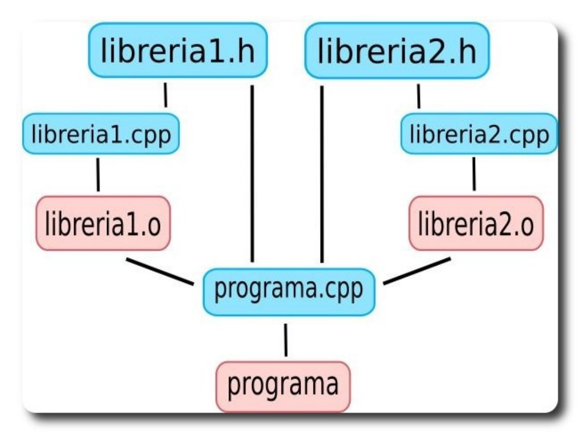
Para cumplir esta tarea, es necesario establecer un pequeño script en un archivo que debe ser nombrado Makefile, y ejecutar el comando make. La principal tarea de Make es comprobar, mediante reglas de dependencia, si los archivos fuente han cambiado desde la última vez que se hizo todo el proceso, para volver a realizarlo de nuevo.

La sintaxis general de un archivo Makefile es la siguiente :

```
objetivo: dependencias...
comandos....
```

Donde objetivo es el archivo a ser creado, dependencias es una lista de un archivos que deben existir antes de crear el objetivo. Finalmente en comandos se especifica cómo crear el objetivo en base a las dependencias. Los comandos pertenecen al sistema operativo , y no solo están limitados a compilar, se pueden utilizar todas las herramientas de línea de comandos del sistema operativo, y no existe una limitación de las cosas que se pueden realizar en este aspecto.

El siguiente ejemplo fue mencionado anteriormente en la primera parte de este tutorial, pero ésta vez utilizaremos Make:



Gráfica de dependencias de este ejemplo.

Cuadros azules son archivos fuente rojos son archivos binarios generados

Para crear un ejemplo lo menos engorroso posible, se omitirá ciertos estándares de codificación, en favor de simpleza. Los archivos fuente se listan a continuación:

libreria1.h

```
int suma(int a, int b);
```

libreria1.cpp

```
#include "libreria1.h"
int suma(int a, int b){
      return (a+b);
}
```

libreria2.h

```
int resta(int a, int b);
```

libreria2.cpp

```
#include "libreria2.h"
int resta(int a, int b){
      return (a-b);
}
```

programa.cpp

```
#include "libreria1.h"
#include "libreria2.h"
#include <iostream>
using namespace std;
int main(void){
      int a = suma(1,2);
      int b = resta(4,3);
      cout << " La suma es: "
            <<" y la resta es: " << b
            << endl;
      return 0;
}
```

Una vez escritos todos los archivos fuente, se debe crear un fichero denominado Makefile, donde se especificarán todas las reglas de dependencia necesarias:

```
CC=g++
CFLAGS=-g

all: programa

programa: libreria1.o libreria2.o
programa.cpp libreria1.h libreria2.h
       $(CC) $(CFLAGS) programa.cpp
libreria1.o libreria2.o -o programa

libreria1.o: libreria1.cpp
       $(CC) $(CFLAGS) libreria1.cpp -o
libreria1.o -c

libreria2.o: libreria2.cpp
       $(CC) $(CFLAGS) libreria2.cpp -o
libreria2.o -c

clean:
       rm *.o programa
```

y para construir y generar todo, simplemente es necesario ejecutar make

```
arn@localhost:~$ make
g++ -g libreria1.cpp -o libreria1.o -c
g++ -g libreria2.cpp -o libreria2.o -c
g++ -g programa.cpp libreria1.o
libreria2.o -o programa
```

Si se usa make nuevamente no pasara nada, puesto que el programa detecta que todos los archivos a generar ya existen y no es necesario una recompilación.

```
arn@localhost:~$ make
make: No se hace nada para ‘all’.
```

Probemos borrando el archivo objeto libreria2.o , y luego haciendo make nuevamente

```
arn@localhost:~$ rm libreria2.o
arn@localhost:~$ make
g++ -g libreria2.cpp -o libreria2.o -c
g++ -g programa.cpp libreria1.o
libreria2.o -o programa
```

Como pueden observar Make detecta que libreria2.o no existe y es generada nuevamente. Como programa depende de librería también es recompilado.

Tomar en cuenta que libreria1.o no fue recompilado esta vez, porque no fue necesario.

Dentro del Makefile también especificamos la regla de construcción denominada clean, que borrará todos los archivos generados durante el proceso, es decir, limpiará el directorio. Para usarlo se debe utilizar make clean :

```
arn@localhost:~$ make clean
rm *.o programa
```

Hasta aquí llega la segunda parte de este tutorial, espero que lo hayan disfrutado. Aún queda una tercera parte por venir, que incluye las herramientas de automatización autotools

Referencias

[1] http://sources.redhat.com/autobook

[2] http://www.gnu.org/prep/standards/

Autor



Arnold Marcelo Guzmán
Desarrollador
spacerockganimedes@gmail.com

Introducción a Django (4ra parte)

Django es un framework para el desarrollo de aplicaciones Web desarrollado en Python originalmente por Adrian Holovaty, Simon Wilson, Jacob Kaplan-Moss y Wilson Miner para World-Online el 2003 . Desde 2005 es software de código abierto (con una licencia BSD) y en septiembre de 2008 alcanzará la tan ansiada versión 1.0.

Introducción

En esta cuarta entrega de la serie vamos a configurar más adecuadamente la interfaz de administración, vamos a utilizar algunas de las vistas genéricas y vamos a crear una vista para extraer artículos de la base de datos. Finalmente desarrollaremos las plantillas para la presentación de los artículos y las categorías de nuestra aplicación.

Configurando la interfaz de administración.

Vamos a empezar configurando de forma más adecuada la interfaz de administración de nuestra aplicación, para mejorar la usabilidad, añadiendo una búsqueda simple a la interfaz, “filtros” para las categorías y un mejor widget para seleccionarlas. Además vamos a llenar automáticamente el campo slug.

Todo esto lo hacemos simplemente definiendo una clase por modelo que queremos configurar, derivada de la clase admin.ModelAdmin.

atix/articles/admin.py

```
from django.contrib import admin

from models import Article, Category

class ArticleAdmin(admin.ModelAdmin):
    filter_horizontal = ('categories',)
    list_display = ('title', 'published', 'last_updated')
    list_filter = ('categories',)
    prepopulated_fields = {'slug': ('title',)}
    search_fields = ('title', 'intro', 'body')

class CategoryAdmin(admin.ModelAdmin):
    list_display = ('name', 'slug',)
    prepopulated_fields = {'slug': ('name',)}

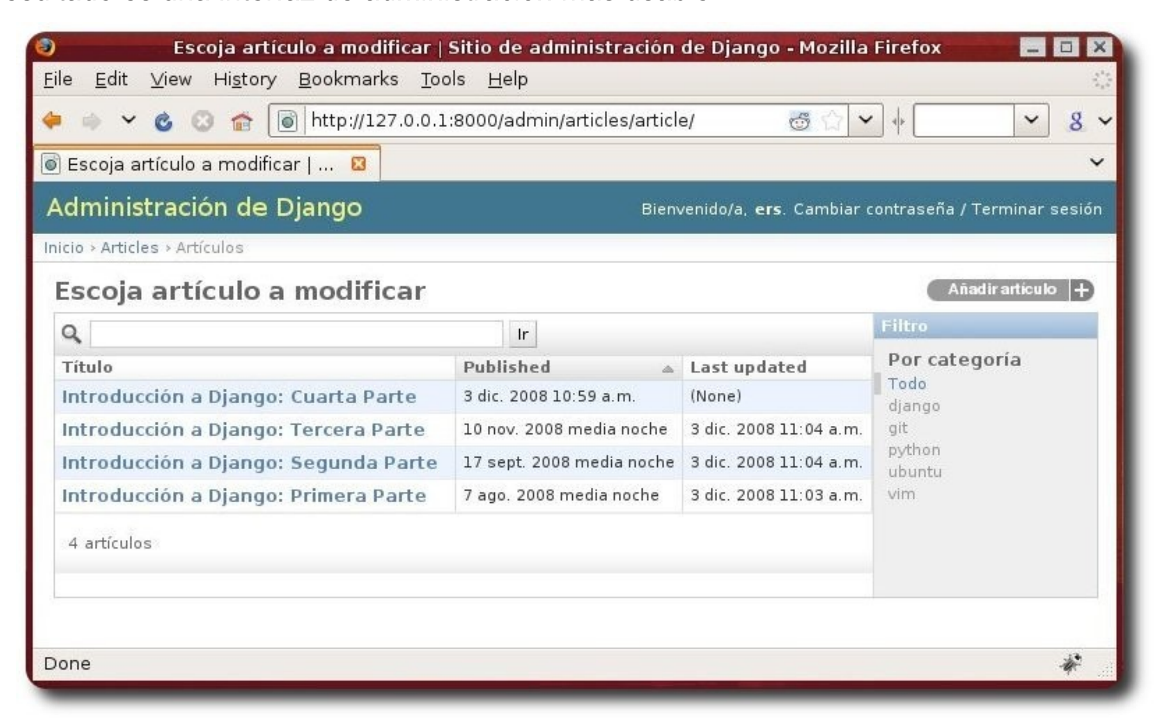
admin.site.register(Article, ArticleAdmin)
admin.site.register(Category, CategoryAdmin)
```

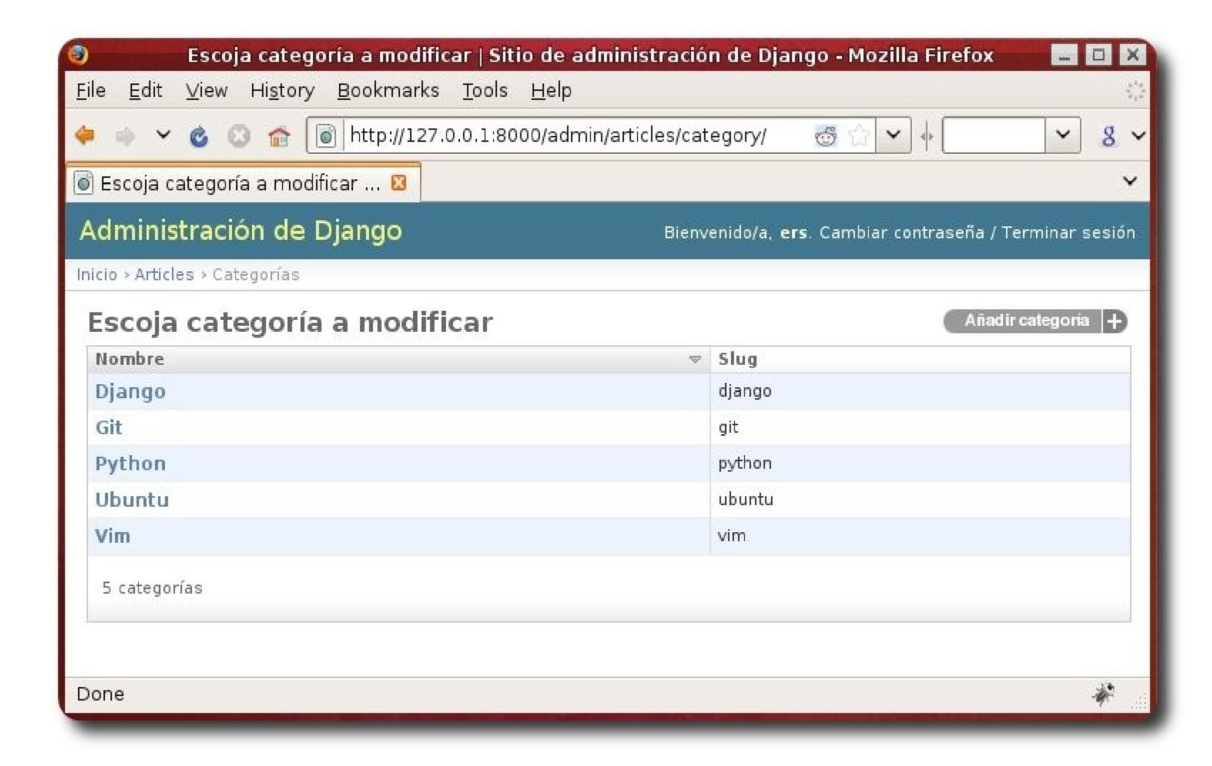
- ✔ Con la opción **`filter_horizontal`** creamos un widget más adecuado a la selección de categorías para nuestros artículos, existe también la opción **`filter_vertical`**, para crear un widget similar de apariencia vertical.
- ✔ Con la opción **`list_display`** definimos que campos de nuestro modelo se van a mostrar en la lista de artículos y con la opción **`list_filter`** definimos los campos que van a servir para agrupar a los artículos.

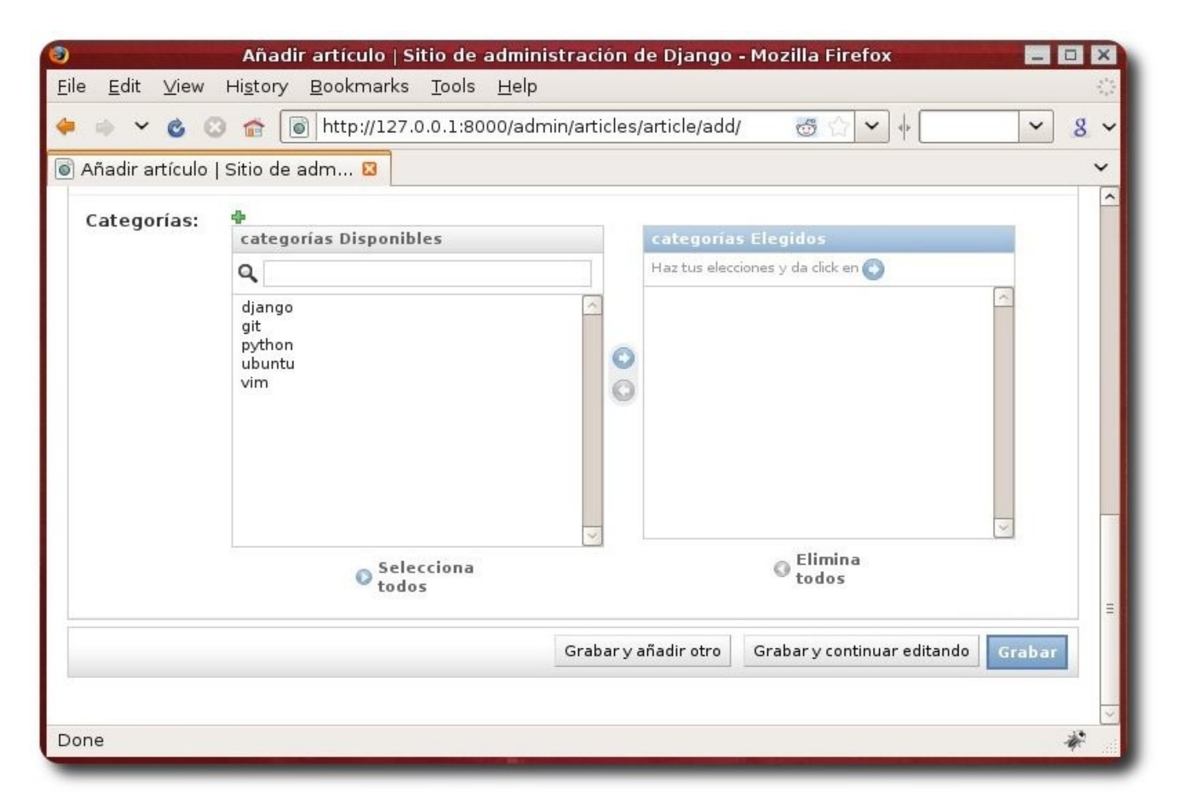
- ✔ Con la opción **`prepopulated_fields`** definimos que el campo slug va a ser llenado automáticamente en base al título del artículo y el nombre de la categoría.
- ✔ Con la opción **`search_fields`** definimos un campo de búsqueda simple en base a algunos de los campos.

Finalmente registramos las opciones adicionales, invocando admin.site.register() con el modelo y la clase de opciones que acabamos de definir.

El resultado es una interfaz de administración más usable.







Vistas genéricas en Django.

Una de las grandes ventajas de Django es la existencia de lo que se conoce como vistas genéricas. Éstas nos permiten simplificar una serie de tareas que son de por sí repetitivas como el extraer una lista objetos de objetos o un objeto en particular de la base de datos.

Django ofrece cuatro categorías de vistas genéricas:

- ✔ Simples: **`django.views.generic.simple`**.
- ✔ De archivo por fecha: **`django.views.generic.date_based`**.
- ✔ De lista y detalle: **`django.views.generic.list_detail`**.
- ✔ De creación, actualización y borrado: **`django.views.generic.create_update`**.

En esta ocasión nos vamos a concentrar en las vistas de lista y detalle y vamos a usar la vista **`object_list`** para extraer las listas de los artículos y de las categorías y **`object_detail`** para extraer el detalle de una categoría.

Para utilizar las vistas solo necesitamos establecer mediante el despacho, que veremos a continuación, que vistas (genéricas) vamos a emplear y pasarles un par de argumentos.

La vista **`django.views.generic.list_detail.object_list`** requiere como argumento queryset, el conjunto de objetos de la base de datos y tiene entre los argumentos opcionales: **`template_object_name`**, que nos permite cambiar el nombre del objeto en la plantilla. El nombre es por defecto object, y la lista de objetos es **`object_list`**.

La vista **`django.views.generic.list_detail.object_detail`** requiere como argumentos queryset, el conjunto de objetos de la base de datos y el **`object_id`** o slug del objeto que nos interesa. Tiene entre los argumentos opcionales **`template_object_name`**, que nos permite cambiar el nombre del objeto en la plantilla. El nombre por defecto es object. Si nos referimos al objeto a través del slug, es necesario también el argumento **`slug_field`**, para indicar cual es el campo slug al que nos referimos en el objeto.

La documentación de referencia de las vistas genéricas está disponible en **http://docs.djangoproject.com/en/dev/ref/generic-views/**

Creando una vista.

Para extraer el detalle de los artículos (un artículo) de la base de datos y presentarlo, podríamos recurrir una vez más a las vistas genéricas, más específicamente a la vista **object_detail** de **django.views.generic.date_based**, pero para aprender como definir una vista, vamos a crear una propia, usando dos "atajos" que nos proporciona Django.

Una vista es una función cuyo primer parámetro es request, que encapsula al pedido HTTP, el resto de los parámetros son los que forman parte del pedido y que han sido extraídos a través de la expresión regular correspondiente en el mapeo de despacho (urls.py) que veremos a continuación.

atix/articles/views.py

```
from django.shortcuts import get_object_or_404, render_to_response

from models import Article

def article_detail(request, year, month, slug):
    article = get_object_or_404(Article,
        published__year=year,
        published__month=month,
        slug=slug)

    return render_to_response('articles/article_detail.html',
        {'article': article})
```

La función **get_object_or_404** intenta extraer de la base de datos un objeto que corresponda a los parámetros que recibe, en este caso se busca un objeto de la clase Article cuyo mes y año de publicación, así como el slug correspondan a los proporcionados como parámetros.

En caso de que no exista dicho objeto, la función elevará una excepción Http404, ocasionando un error 404, mostrando la página de error de la plantilla **atix/templates/404.html**. Si existiese más de un objeto con esas características el resultado también será un error.

La función **render_to_response** devolverá como respuesta el resultado de la evaluación de la plantilla **atix/templates/articles/article_detail.html,** con el contexto article, es decir con el artículo que acabamos de extraer de la base de datos como única variable.

Despacho: **urls.py.**

El despacho establece la relación entre las rutas el el servidor Web (URLs) y las vistas que se van a emplear. En Django, el despacho se configura en base a expresiones regulares.

atix/urls.py

```
from django.conf.urls.defaults import *

from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    (r'^admin/(.*)', admin.site.root),
    (r'^contact/', include('contact_form.urls')),
    (r'^articles/', include('articles.urls')),
)
```

En nuestro caso añadimos una línea a la configuración global, estableciendo que toda ruta con la raíz **/articles/** estará a cargo la configuración de la aplicación en un archivo propio de configuración.

atix/articles/urls.py

```
from django.conf.urls.defaults import *
from django.views.generic.list_detail import object_detail, object_list

from models import Article, Category
from views import article_detail

article_info = {
    'queryset': Article.objects.all(),
    'template_object_name': 'article',
}

category_info = {
    'queryset': Category.objects.all(),
    'template_object_name': 'category',
}

urlpatterns = patterns('',
    url(r'^$',
        object_list,
        article_info,
        name='article-list'),
    url(r'^categories/$',
        object_list,
        category_info,
        name='category-list'),
    url(r'^category/(?P<slug>[\w-]+)/$',
        object_detail,
        dict(category_info, slug_field='slug'),
        name='category-detail'),
    url(r'^(?P<year>\d{4})/(?P<month>\d{1,2})/(?P<slug>[\w-]+)/$',
        article_detail,
        name='article-detail'),
)
```

Los diccionarios **article_info** y **category_info** fijan los argumentos descritos anteriormente.

Los patrones de ruta url() tienen los siguientes parámetros:

- ✔ la expresión regular a la que corresponden,
- ✔ la vista encargada de procesar el pedido y generar una respuesta,
- ✔ los argumentos de la vista, si ésta los requiere y, opcionalmente,
- ✔ el nombre de la ruta.

Las dos primeras rutas hacen uso directo de la vista genérica **object_list** y los argumentos definidos para mostrar una lista de artículos (bajo la ruta **/articles/**) y una lista de categorías (bajo la ruta **/articles/categories/**).

La tercera ruta establece que la vista genérica **object_detail** y los argumentos definidos, añadiendo **slug_field**, serán usados para mostrar el detalle de una categoría. El slug será una cadena de uno o más caracteres o un guión (“-”), bajo la ruta **/articles/category/**.

La cuarta ruta establece que la vista **article_detail** que definimos anteriormente será la encargada de mostrar el detalle de un artículo. La vista recibirá directamente year, month y slug como parámetros. El año serán cuatro dígitos, el mes uno o dos y el slug una vez más una cadena de uno o más caracteres o un guión (“-”), bajo la ruta **/articles/**.

Plantillas para los artículos y las categorías.

atix/templates/articles/article_list.html

```
{% extends "base.html" %}

{% block title %}Artículos | {{ block.super }}{% endblock %}

{% block content %}
<h2>Artículos</h2>

{% for article in article_list %}
<h3>
  <a href="{{ article.get_absolute_url }}">{{ article.title }}</a>
  ({{ article.published|date:"F Y"|title }})
</h3>

{{ article.intro|safe }}
{% endfor %}
{% endblock %}
```

En la lista de artículos iteramos por la lista que recibe la plantilla de la vista y mostramos para cada uno la introducción, marcando a través del filtro safe que su contenido es seguro, es decir puede contener etiquetas HTML.

atix/templates/articles/article_detail.html

```
{% extends "base.html" %}

{% block title %}{{ article.title }} | {{ block.super}}{% endblock %}

{% block content %}
<h2>{{ article.title }}</h2>

<h3>{{ article.published|date:"j F Y"|title }}</h3>

{{ article.intro|safe }}

<hr noshade>

{{ article.body|safe }}
{% endblock %}
```

En el detalle de un artículo mostramos la introducción y el cuerpo del artículo, marcando nuevamente su contenido como seguro.

atix/templates/articles/category_list.html

```
{% extends "base.html" %}

{% block title %}Categorías | {{ block.super }}{% endblock %}

{% block content %}
<h2>Categorías</h2>

{% for category in category_list %}
<h3>
  <a href="{{ category.get_absolute_url }}">{{ category.name }}</a>
  {{ category.articles.count }}
  artículo{{ category.articles.count|pluralize }}
</h3>
{% endfor %}
{% endblock %}
```

En la lista de categorías iteramos por la lista que recibe la plantilla de la vista y mostramos para cada uno el nombre y el número de artículos publicados. El filtro pluralize añade una “s” si el valor de **`category.articles.count`** es mayor a uno.

atix/templates/articles/category_detail.html

```
{% extends "base.html" %}

{% block title %}
  Artículos en la categoría &raquo;{{ category.name }}&laquo; |
  {{ block.super}}
{% endblock %}

{% block content %}
<h2>
  Artículos en la categoría &raquo;{{ category.name }}&laquo;
</h2>

{% for article in category.articles.all %}
<h3>
  <a href="{{ article.get_absolute_url }}">{{ article.title }}</a>
  ({{ article.published|date:"F Y"|title }})
</h3>
{% endfor %}
{% endblock %}
```

En el detalle de una categoría iteramos por la lista de artículos publicados en esa categoría para mostrar el título y la fecha de publicación.

Artículos | atix - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://127.0.0.1:8000/articles/

Artículos | atix

Artículos

Introducción a Django: Cuarta Parte (Diciembre 2008)

En esta cuarta entrega de la serie vamos a configurar más adecuadamente la interfaz de administración, vamos a utilizar algunas de las vistas genéricas y vamos a crear una vista para extraer artículos de la base de datos. Finalmente desarrollaremos las plantillas para la presentación de los artículos y las categorías de nuestra aplicación.

Introducción a Django: Tercera Parte (Noviembre 2008)

En esta tercera entrega de la serie vamos a crear una primera aplicación que nos permitirá publicar artículos. Vamos a definir nuestros modelos, utilizando algunos de los campos que nos ofrece Django y algunas de las relaciones básicas y vamos a activar la interfaz de administración para nuestra nueva aplicación.

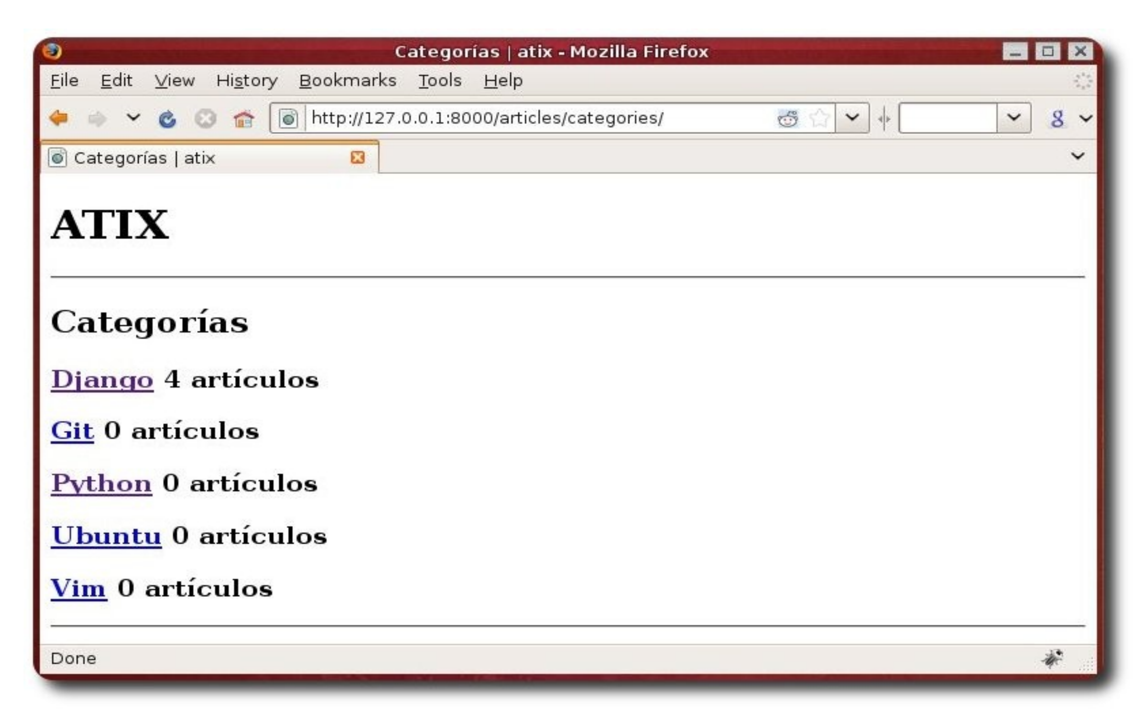
Introducción a Django: Segunda Parte (Septiembre 2008)

En esta entrega vamos a integrar una aplicación adicional, contact_form, para tener un formulario de contacto en nuestro sitio y vamos a aprender más sobre las plantillas en Django.

Introducción a Django: Primera Parte (Agosto 2008)

En esta serie vamos a desarrollar paso a paso un sitio completo con Django, y vamos a describir cada uno de los componentes principales en el camino. Inicialmente vamos a crear un sitio con lo más básico: páginas estáticas. Para ello vamos a ver cómo se instala Django, cómo se crea y configura un proyecto y cómo activamos una aplicación para tener un páginas estáticas.

Done



En la próxima entrega cerraremos la serie añadiendo feeds de sindicación (RSS y Atom) para los artículos publicados, añadiremos más funcionalidad a la interfaz de administración y prepararemos nuestro proyecto para un entorno de producción.

Referencias

[1] http://www.djangoproject.com/

Autor



Ernesto Rico Schmidt
Usuario de Linux y Software Libre desde 1994
e.rico.schmidt@gmail.com

Introducción a Ext JS (2da parte)

Ext JS es un Framework que permite crear aplicaciones Web 2.0 con interfaces muy similares a la de una aplicación de escritorio.

Introducción

En la primera parte de este tutor (número 5 de la revista ATIX), dimos una introducción a Ext JS un framework de desarrollo en JavaScript que se encarga de la parte del cliente, ahora vamos a estudiar el uso y manejo de árboles, no solo veremos la parte del cliente sino también como se integra en el servidor, donde haremos uso de PHP.

Los árboles son estructuras jerárquicas que se constituyen de las raíces, las ramas y las hojas, las raíces son el primer nodo del árbol del cual se cuelgan todos los demás y que no tienen dependencia de otro nodo es decir no tienen padre, las ramas son los hijos y nodos intermedios entre la raíz y las hojas, las hojas son nodos finales los cuales no tienen hijos, un ejemplo muy común de árboles es el sistema de archivos de un sistema operativo donde las carpetas conforman las raíces, ramas y hojas.

Iniciando el trabajo de árboles

Primero debemos crear la base de datos test y la tabla árbol, para crearla haremos uso del siguiente SQL.

```
CREATE TABLE IF NOT EXISTS `arbol` (
  `id` int(11) NOT NULL auto_increment,
  `parent_id` int(11) default NULL,
  `nombre` varchar(50) character set utf8 collate utf8_unicode_ci NOT NULL,
  PRIMARY KEY  (`id`)
) ENGINE=MyISAM;
```

Ahora debemos crear tres scripts, creen primero la carpeta del proyecto árbol, dentro de esta carpeta descompriman la librería Ext JS, ahora deben crear tres scripts **`index.php`**, **`db.php`** y **`arbol.php`**, cabe recalcar que el código está escrito para PHP5. Ahora abran en su editor el archivo **`index.php`** y adjunten el siguiente código .

`index.php`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" /><title>Arboles con
Ext JS</title>
<link rel="stylesheet" type="text/css" href="ext-2.2/resources/css/ext-all.css" />
<style type="text/css">
body{
    height:100%;
    font-family:georgia;
}
```

```
</style>
<script type="text/javascript" src="ext-2.2/adapter/ext/ext-base.js"></script>
<script type="text/javascript" src="ext-2.2/ext-all-debug.js"></script>
<!--Contenido que sera modificado-->
<!-- Fin -->
<body>
    <div id="arbol" ></div>
</body>
</html>
```

Lo que hacemos aquí es básicamente llamar a la hoja de estilos así como los scripts necesarios para que pueda funcionar Ext JS, ahora ejecutemos el siguiente SQL en nuestro gestor de base de datos preferido.

```
INSERT INTO `arbol` (`id`, `parent_id`, `nombre`) VALUES
(1, NULL, 'Frameworks'),
(2, 1, 'Ruby on Rails'),
(3, 1, 'Django'),
(4, 1, 'Symfony'),
(5, 1, 'CakePHP');
```

Con esto hemos creado un Padre que es Frameworks y varios hijos, ahora modifiquemos el archivo

index.php

```
<!--Contenido que sera modificado-->
<script>

Ext.BLANK_IMAGE_URL = 'ext-2.2/resources/images/default/s.gif';

Ext.onReady(function() {

    //Crear un nodo que cargue asicronicamente (AJAX) sus hijos
    root = new Ext.tree.AsyncTreeNode({
        text: 'Raiz',
        id:'0',
        draggable: false
    });

    //Crear Arbol

    var tree = new Ext.tree.TreePanel({
        id: 'treePanel',
        loader: new Ext.tree.TreeLoader({
            url:'arbol.php',
            requestMethod:'GET',
            baseParams:{accion:'mostrar'}
        }),
        width: 250,
        height: 300,
        enableDD: true, //Permite Drag and Drop
        containerScroll: true,
        renderTo: 'arbol', //Id del tag en el cual se renderiza
        root: root,
        rootVisible: false, //No queremos ver el nodo raiz
        /*tbar : [{
            text: "Crear", handler: createNode},
            {text: "Borrar", handler: deleteNode
        }],*/
```

```
        //Definición de eventos
        listeners: {
            //Se define el nodo actual al que se haya seleccionado para poder crear
            //hijos a partir del nodo seleccionado
            click: {fn: function(node) { currentNode = node} }
            //beforeappend: {fn: function() {currentNode.expand() } }
        }
      });
    root.expand();
});
</script>
<!-- Fin -->
```

Es turno de modificar el archivo:

arbol.php

```
<?php
include_once('db.php');
class Arbol{
       var $datos, $bd;
       function __construct($data, $bd) {
              $params = $data;
              unset($params['accion']);
              $this->bd = $bd;
              $this->{$data['accion']}($params);
       }
       // Presenta los hijos de un padre
       function mostrar($params) {
              $buscar = "={$params['node']}";
              if($params['node'] == 0) {
                     $buscar = " IS NULL";
              }
              $res = mysql_query("select * from arbol where parent_id".$buscar, $this->bd);
              $array = array();
              while ($fila = mysql_fetch_assoc($res)) {
                     $array[] = array('id' => $fila['id'], 'text' => $fila['nombre'], 'leaf' => false);
              }
              $this->presentar($array);
       }
       function presentar($data) {
              echo json_encode($data);
       }
}
if(isset($_REQUEST)) {
       $arbol = new Arbol($_REQUEST, $link);
}
```

Para realizar la conexión a la base de datos, debemos modificar el archivo:

db.php

```
<?php
$link = mysql_connect('localhost', 'demo', 'demo');
if (!$link) {
   die('Not connected : ' . mysql_error());
}
// make foo the current db
$db_selected = mysql_select_db('test', $link);
if (!$db_selected) {
   die ('Can\'t use test: ' . mysql_error());
}
```

Como pueden ver ahora cuando hacemos click en los nodos este llamará a la función mostrar en la clase Arbol y presentará la lista de nodos hijos que tiene este nodo, ahora adicionemos más nodos en nuestra base de datos, mediante el siguiente SQL

```
INSERT INTO `arbol` (`id`, `parent_id`, `nombre`) VALUES
(6, 2, 'Modelo'), (7, 2, 'Vista'), (8, 2, 'Controlador'),(9, 3, 'Rad, hecho en
Python');
```

Ahora lo que hemos hecho es adicionar nuevos nodos a los Frameworks Ruby on Rails y Django, cuando ustedes hagan click en estos nodos verán que se expanden y muestran sus hijos.

Ahora hagamos algo más, cuando movamos un nodo lo vamos a reemparentar, es decir vamos a cambiar de padre, lo que sucede en este tipo de casos es que uno selecciona un nodo y lo mueve a otro padre, dicho de otra forma soporta el comportamiento *Drag and Drop*, esto hace de que se ejecute el evento mover movenode, el cual automáticamente nos pasa los parámetros necesarios para poder realizar y enviar estos datos al servidor.





Comencemos primero modificando el archivo **arbol.php** y adicionemos el siguiente código a la clase Arbol .

```
function mover($params) {
             $query = "update arbol set parent_id={$params['nuevoPadre']} where
id={$params['nodo']}";
             if (mysql_query($query, $this->bd)) {
                    $this->presentar(array('success' => true));
             }else{
                    $this->presentar(array('success' => false));
             }
       }
```

Lo que básicamente sucede es que esta función actualiza los datos del nuevo padre, pero la parte más simple sucede en el servidor, para que este comportamiento sea posible hay que modificar el archivo **index.php** y aumentar el siguiente código después del bloque **var tree = new Ext.tree.TreePanel({...})**;

```
/**
    *Evento que permite realizar el movimiento de un nodo
    */
    tree.on('movenode', function(arbol, node, oldParent, newParent, position) {

        if(newParent.id == oldParent.id) {
            //No realizar nada no se reemparento
            return false;
        }

        //Inicio de llamada al servidor
        Ext.Ajax.request({
            url:'arbol.php',
            method: 'GET',
            params: {accion: 'mover', nodo: node.id, nuevoPadre: newParent.id},
            success: function(resp, o) {
              try{
                //Decodes JSON
                r = Ext.decode(resp.responseText);
                //Verificar respuesta exitosa
                if(!r.success) {
                  o.failure(); //Fallo
                }
                arbol.enable(); //Habilitar Panel
              //Capturar excepcion
              }catch(e) {
                o.failure();
              }
            },
            //Funcion de falla
            failure: function(resp, o) {

              arbol.suspendEvents();
              oldParent.insertBefore(node, null);
              arbol.enable(); //habilitar arbol
            }
        });
    });
```

Se captura el evento **movenode** que pasa varios parámetros: árbol, **node, oldParent, newParent, position**, primero verificamos de que el nuevo padre no sea el mismo que el antiguo padre, ya que se puede mover el nodo pero soltarlo en el mismo padre, una vez verificado iniciamos una llamada Ajax a la cual le pasamos los parámetros: **params: {accion: 'mover', nodo: node.id, nuevoPadre: newParent.id}**, los cuales indican el nodo que se mueve, el nuevo padre y la acción a ejecutarse, en caso de que las respuesta sea correcta la

respuesta del servidor es: `{success: true}` codificada en JSON, en caso de falla se llama a la función failure, dentro de la llamada Ajax y se retorna al nodo al padre original.

En la siguiente versión de este tutor continuaremos con este mismo árbol solo que esta vez vamos a modificar el nombre, crear nuevos nodos y borrar, espero les haya gustado el tutor, pueden encontrar una versión similar para Ruby on Rails de este tutor en **`http://boliviaonrails.com`**.

Referencias

[1] http://extjs.com

[2] http://getfirebug.com

Autor



Boris Barroso
boriscyber@gmail.com

Desarrollo Ágil con Ruby on Rails (2da Parte)

Ruby on Rails es un framework de desarrollo web ágil, elaborado por David Heinemeier Hansson, que el 2004 lanzó la versión pública a partir de desarrollar el proyecto Basecamp, Ruby on Rails (RoR) está desarrollado en el lenguaje de programación Ruby, RoR actualmente se encuentra en la versión 2.1.1 en la que contribuyeron más de 1400 desarrolladores del mundo con 1600 parches al framework, por estos datos RoR es uno de los proyectos open source con más movimiento actual.



Introducción

Este 21 de Noviembre, salió a la luz la versión 2.2.2 estable de rails, la que incorpora distintas características como: Thread Safety , Internacionalización i18n dentro del core, Migraciones transaccionales, y otras características, además salió la versión definitiva de Netbeans 6.5, daremos tips para actualizar el proyecto a la nueva versión del framework.

Continuando con el desarrollo de la aplicación Atix, en esta segunda parte primeramente explicaré como actualizar el proyecto a la nueva versión de rails, además veremos validaciones de campos, manejo de relaciones de tablas.

Actualizar a la última versión de Rails.

Aclaración. Esto no es realmente necesario, en ambientes de producción, ya que se sugiere hacer pruebas si realmente el proyecto está listo para la actualización, ya que llevaría a que la aplicación no esté disponible y causar desagradables momentos, con esta advertencia y para propósitos demostrativos actualizaremos el proyecto.

Antes tienen que instalar/actualizar rubygems a la versión 1.3.1 y ejecutar:

```
(sudo) gem install rails
```

Con esto actualizaremos a la última versión de rails en la máquina, ahora lo realizaremos en el proyecto, dentro del archivo **`config/envinroment.rb`** en la primera línea encontraremos la línea:

```
RAILS_GEM_VERSION = '2.1.1' unless
defined? RAILS_GEM_VERSION
```

Que la cambiamos por:

```
RAILS_GEM_VERSION = '2.2.2' unless
defined? RAILS_GEM_VERSION
```

Ahora realizamos la tarea **`rake: rails:update`** en consola, para netbeans: seleccionando el proyecto y apretando Alt + Shift + R te saldrán las tareas rake disponibles y seleccionamos **`rails:update`**, como último paso tendremos que ir al archivo **`config/database.yml`** añadimos el número de conexiones simultáneas que accederán a la aplicación, anteriormente en rails una sola conexión accedía a la aplicación, lo que permitía que a veces la aplicación estuviera fuera de disposición por realizar una consulta pesada, para eso añadimos una línea:

```
development:
  adapter: mysql
  encoding: utf8
  database: atix_development
  pool: 5
  username: root
  password:
  host: localhost
```

Con esto indicamos que 5 hilos accederán a la base de datos, 5 es por defecto puedes cambiarlo de acuerdo a tus necesidades esto repetiremos a las demás envinroments definidas en **database.yml**.

Así ya hemos actualizado a la última versión de rails.

Validaciones y relación de tablas.

Para la clase revista tenemos que validar que siempre se llene un nombre descriptivo y su editorial, luego el el campo número debe ser numérico y no aceptar otros formatos, en relaciones cada revista tiene varios artículos, el código para el modelo sería:

```
class Revista < ActiveRecord::Base
  validates_presence_of :nombre,
:editorial
  validates_numericality_of :numero
  has_many :articulos
end
```

Ahora para la clase artículos, cada artículo debe de tener llenado su título y su contenido y además las relaciones son del tipo: un artículo tiene una revista y este artículo tiene un único autor, entonces el código es:

```
class Articulo < ActiveRecord::Base
  validates_presence_of :titulo,
:contenido
  belongs_to :revista
  belongs_to :autor
end
```

Y por último la clase autor debe de tener llenados forzosamente su nombre, apellidos y su email, además que cada autor puede tener escritos cero o más artículos, lo cual es:

```
class Autor < ActiveRecord::Base
  validates_presence_of :nombre,
:apellidos, :email
  has_many :articulos
end
```

Explicaré que es lo que realiza las acciones **has_many :articulos**, en el caso de autor, le dice a la clase Autor que cree un método llamado artículos que devuelva todos los registros que concuerden con este autor, para esto busca en la tabla Artículos mediante la llave foránea revista_id (esto lo hace automáticamente).

Para el caso de **belongs_to :revista**, crea un método dentro de Articulo, llamado revista en el que devuelve un único registro de la tabla Revista, buscando por el campo revista_id de Artículo.

Personalizando Vistas

Ahora modificaremos un poco las vistas para que reflejen los cambios:

/app/views/articulos/index.html.erb

```
<h1>Listado de articulos</h1>
<table>
  <tr>
    <th>Titulo</th>
    <th>Contenido</th>
    <th>Revista</th>
    <th>Autor</th>
    <th colspan="3">Acciones</th>
  </tr>
```

```
<% for articulo in @articulos %>
  <tr>
    <td><%=h articulo.titulo %></td>
    <td><%=h articulo.contenido %></td>
    <td><%= link_to(articulo.revista.nombre, articulo.revista) if articulo.autor
%></td>
    <td><%= link_to(articulo.autor.nombre, articulo.autor) if articulo.autor %></td>
    <td><%= link_to 'Mostrar', articulo %></td>
    <td><%= link_to 'Edit', edit_articulo_path(articulo) %></td>
    <td><%= link_to 'Borrar', articulo, :confirm => 'Esta seguro?', :method => :delete
%></td>
  </tr>
<% end %>
</table>
<br />
<%= link_to 'Nuevo articulo', new_articulo_path %>
```

Aquí lo más importante es la incorporación de los enlaces a la revista y al autor como se ve en la imagen, estos enlaces van al método show respectivo:



Hasta aquí el tutorial, pondré a disposición el código fuente en mi blog **http://ww.carakan.com**, en el próximo número realizaremos trabajo con ajax y paginación de resultados.

Agradecimientos.

Agradezco a **Atix** por el espacio que me brinda, apoyando el software libre en Bolivia, así mismo invito a todo la audiencia a visitar nuestra pagina **http://www.casasmap.com**, desarrollado enteramente en software libre.

Referencias

[1] http://www.rubyonrails.org/

[2] http://www.rubyforge.org/

Autor



Carlos Ramos
Lic. Informática UMSA
Lider de Wiebia, soluciones web 2.0
carakan@gmail.com
Blog personal http://ww.carakan.com
http://www.wiebia.com

Trac: Gestión de proyectos de desarrollo de Software (2da parte)

La gestión de proyectos de desarrollo de software, es un elemento imprescindible al momento de encarar proyectos de desarrollo, porque esto implica considerar tópicos como: control de versiones, wikis, manejo de bugs, etc. En la actualidad existen varias opciones para este fin, pero una de las que destaca en el mundo del software Libre es **Trac,** por su sencillez, facilidad y por su calidad..



Introducción

En la primera parte del artículo habíamos visto los aspectos conceptuales e iniciales de forma general de como hacer uso de **Trac** para la gestión de proyectos de desarrollo de Software, en esta segunda entrega, consideramos el uso y personalización de la herramienta desde la línea de comandos y desde su entorno web.

Configuración de Trac

Las configuraciones de **Trac** se la realiza mediante su archivo de configuración (**`trac.ini`**), que posee una estructura y sintaxis bastante amigable y comprensible. Este archivo puede ser modificado desde la línea de comandos o por medio del plugin de administración vía web.

Gestión de Trac desde línea de comandos

Trac tiene la posibilidad de ser configurado personalizado y gestionado tanto desde la línea de comandos (modo interactivo), como desde su interfaz web (gracias al plugin WebAdmin) que será visto en la siguiente sección.

En la mayoría de los casos no es preciso memorizar todos los comandos que posee **Trac**, ya que para esto contamos con una ayuda en línea, accesible mediante:

```
trac-admin help
```

Esta nos permitirá listar todas las opciones disponibles, su sintaxis y descripción de las mismas.

La ejecución de comandos puede ser:

- ✔ de modo directo
- ✔ de modo interactivo.

Modo directo

La forma de acceder al modo directo es:

```
trac-admin nombre_proyecto
nombre_del_comando
```

```
jesaavedra@desarrollo:~/trac
Archivo  Editar  Ver  Terminal  Solapas  Ayuda
[jesaavedra@desarrollo trac]$ trac-admin trabajo wiki list

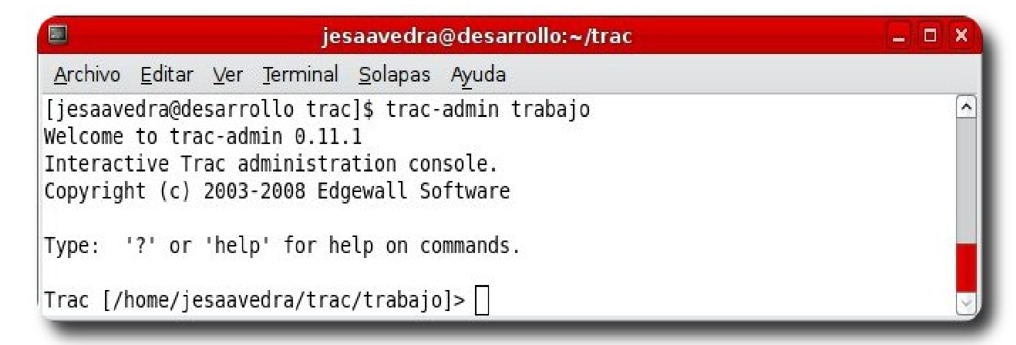
Title                        Edits  Modified
------------------------------------------------------
CamelCase                    1      2008-09-10 11:46:15
InterMapTxt                  1      2008-09-10 11:46:15
InterTrac                    1      2008-09-10 11:46:15
InterWiki                    1      2008-09-10 11:46:15
PageTemplates                1      2008-09-10 11:46:15
RecentChanges                1      2008-09-10 11:46:15
SandBox                      1      2008-09-10 11:46:15
TitleIndex                   1      2008-09-10 11:46:15
TracAccessibility            1      2008-09-10 11:46:15
TracAdmin                    1      2008-09-10 11:46:15
TracBackup                   1      2008-09-10 11:46:15
TracBrowser                  1      2008-09-10 11:46:15
TracCgi                      1      2008-09-10 11:46:15
TracChangeset                1      2008-09-10 11:46:15
TracEnvironment              1      2008-09-10 11:46:15
TracFastCgi                  1      2008-09-10 11:46:15
TracFineGrainedPermissions   1      2008-09-10 11:46:15
TracGuide                    1      2008-09-10 11:46:15
TracImport                   1      2008-09-10 11:46:15
TracIni                      1      2008-09-10 11:46:15
```

Línea de comandos en modo directo

Esto permitirá ver un listado de los wikis disponibles para el proyecto “trabajo” (proyecto creado en la anterior entrega), cuya salida la observamos en el siguiente gráfico.

Modo interactivo

A diferencia del anterior, este nos permite ingresar a un entorno interactivo, donde sólo precisamos introducir los comandos necesarios (entorno parecido al de Python), para acceder a este modo debemos hacerlo digitando **`trac-admin trabajo`**.



Línea de comandos en modo interactivo

Instalación de plugins

Las características de la arquitectura de **Trac**, permiten la instalación de plugins, que le permiten adoptar características adicionales y que coadyuvan a su mejor funcionamiento.

Los plugins al ser paquetes desarrollados python, pueden ser instalados de la siguiente forma:

- ✔ Mediante la utilidad easy_install: **`easy_install nombre_del_plugin.egg`**
- ✔ Mediante su código fuente, dentro el directorio donde se ha ha realizado la descompresión del plugin, ejecutar: **`python setup.py install`**

Para nuestro caso ejemplificaremos instalando el plugin WebAdmin que nos permitirá realizar la gestión de **Trac** mediante su interfaz web, la instalación la realizamos de la siguiente forma:

```
easy_install
http://svn.edgewall.com/repos/trac/sandbo
x/webadmin
```

Activar el plugin

Una de tantas cosas que se puede hacer con el archivo de configuración de **Trac** (trac.ini), es poder habilitar o deshabilitar plugins, así:

```
[components]
nombre_plugin.* = enabled
```

para nuestro ejemplo procedemos a habilitar el plugin WebAdmin, para eso añadimos en el archivo **`../trabajo/conf/trac.ini`** lo siguiente:

```
[components]
webadmin.* = enabled
```

Creación de usuarios

Dentro la gestión de proyectos es muy recomendable que todas y cada una de las tareas sean controladas, y que mejor forma de hacerlo si disponemos de usuarios para ello.

Para la creación de usuarios debemos crear un archivo que los contenga, de esta forma:

```
touch /usr/usuarios
htpasswd -m /usr/usuarios jose
```

Básicamente se hace uso de la utilidad htpasswd, este procedimiento podemos realizarlo para todos los usuarios necesarios, sugerimos añadir un par de usuarios a manera de prueba.

Autenticación de usuarios

Actualmente existen diversas formas y medios de autenticación, ya sean provistos por el propio servidor web, una aplicación, mediante el acceso a una base de datos o mediante el acceso a un fichero, en nuestro caso haremos uso del fichero de usuarios utilizado cuando en la sección anterior, para esto debemos añadir lo siguiente al archivo de configuración de **Trac**:

```
[account-manager]
password_format = htpasswd
password_store = HtPasswdStore
password_file = /aplic/usr/usuarios
```

Privilegios de usuarios

Trac dispone de una serie de privilegios, que nos permiten de cierta forma limitar o permitir el acceso a ciertas partes o acciones dentro de la gestión de nuestro proyecto, como mencionamos anteriormente estas tareas administrativas pueden ser realizadas ya sea desde la línea de comandos o desde la interfaz web (plugin WebAdmin), a continuación vemos algunos ejemplos:

Listar los usuarios más los privilegios asignados, y también las opciones de privilegios disponibles.

```
trac-admin trabajo permission list
```

```
jesaavedra@desarrollo:~/trac
Archivo  Editar  Ver  Terminal  Solapas  Ayuda
[jesaavedra@desarrollo trac]$ trac-admin trabajo permission list

User           Action
------------------------------
anonymous      BROWSER_VIEW
anonymous      CHANGESET_VIEW
anonymous      FILE_VIEW
anonymous      LOG_VIEW
anonymous      MILESTONE_VIEW
anonymous      REPORT_SQL_VIEW
anonymous      REPORT_VIEW
anonymous      ROADMAP_VIEW
anonymous      SEARCH_VIEW
anonymous      TICKET_VIEW
anonymous      TIMELINE_VIEW
anonymous      TRAC_ADMIN
anonymous      WIKI_VIEW
authenticated  TICKET_CREATE
authenticated  TICKET_MODIFY
authenticated  WIKI_CREATE
authenticated  WIKI_MODIFY


Available actions:
 BROWSER_VIEW, CHANGESET_VIEW, CONFIG_VIEW, EMAIL_VIEW, FILE_VIEW,
 LOG_VIEW, MILESTONE_ADMIN, MILESTONE_CREATE, MILESTONE_DELETE,
 MILESTONE_MODIFY, MILESTONE_VIEW, PERMISSION_ADMIN, PERMISSION_GRANT,
 PERMISSION_REVOKE, REPORT_ADMIN, REPORT_CREATE, REPORT_DELETE,
 REPORT_MODIFY, REPORT_SQL_VIEW, REPORT_VIEW, ROADMAP_ADMIN, ROADMAP_VIEW,
 SEARCH_VIEW, TICKET_ADMIN, TICKET_APPEND, TICKET_CHGPROP, TICKET_CREATE,
 TICKET_EDIT_CC, TICKET_EDIT_DESCRIPTION, TICKET_MODIFY, TICKET_VIEW,
 TIMELINE_VIEW, TRAC_ADMIN, WIKI_ADMIN, WIKI_CREATE, WIKI_DELETE,
 WIKI_MODIFY, WIKI_VIEW
```

Listado de privilegios

Asignación de privilegios

Asigna ciertos privilegios al usuario juan

```
trac-admin trabajo permission add juan REPORT_DELETE WIKI_CREATE
```

Asigna al usuario esteban el privilegio de administrador. Cabe mencionar que este privilegio permitirá observar en la barra de opciones la opción ADMIN (provista por el plugin WebAdmin)

```
trac-admin trabajo permission add esteban TRAC_ADMIN
```

El quitar uno o todos los privilegios asignados puede realizarse de la forma anterior.

```
trac-admin  trabajo permission remove juan REPORT_DELETE
trac-admin  trabajo permission remove juan *
```

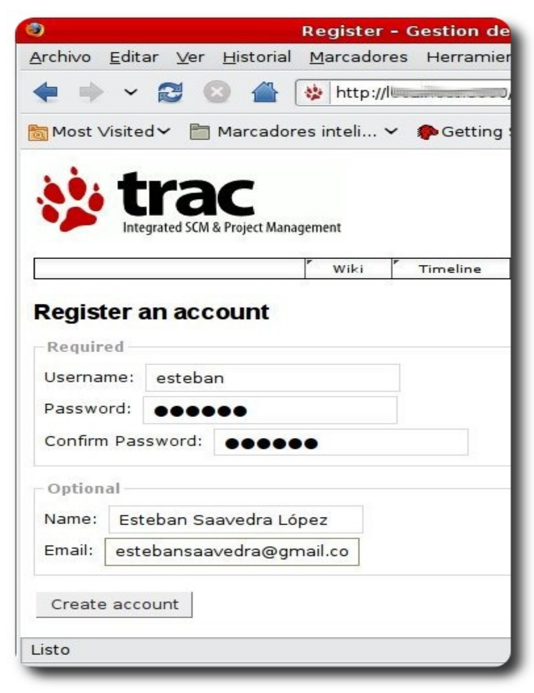
Esta última secuencia de privilegios permite asignar ciertos privilegios al grupo desarrollo y posteriormente asignar miembros a este grupo.

```
trac-admin trabajo permission add desarrollo WIKI_ADMIN
trac-admin trabajo permission add desarrollo REPORT_ADMIN
trac-admin trabajo permission add desarrollo TICKET_MODIFY
trac-admin trabajo permission add juan desarrollo
trac-admin trabajo permission add jose desarrollo
```

Gestión de Trac desde su interface web

Trac por medio del plugin WebAdmin permite realizar la gestión de nuestro proyecto, adicionalmente podemos instalar mediante esta interfaz o desde línea de comandos el plugin AccountManager, que nos brinda la facilidad de también gestionar los usuarios y privilegios de nuestro proyecto (obviamente también podemos instalar todos los plugins que en cierto instante precisemos).

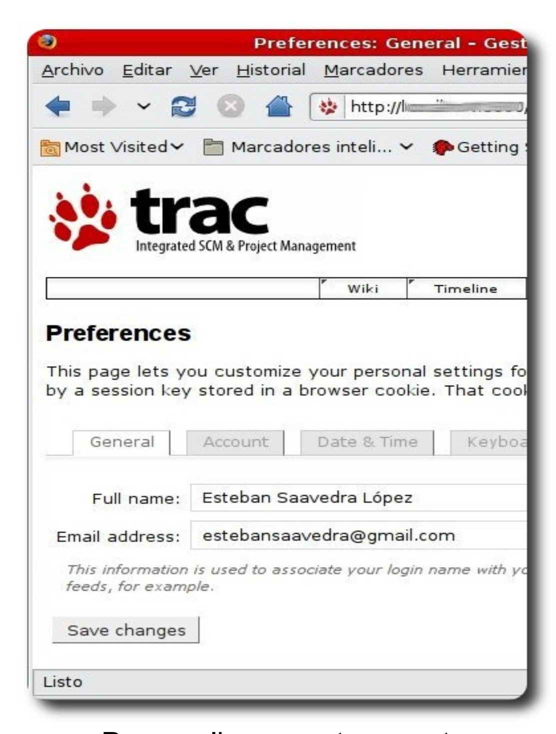
A continuación mostramos algunas capturas que ilustran el uso de estas interfaces.



Registro de una nueva cuenta



Login de una cuenta



Personalizar nuestra cuenta

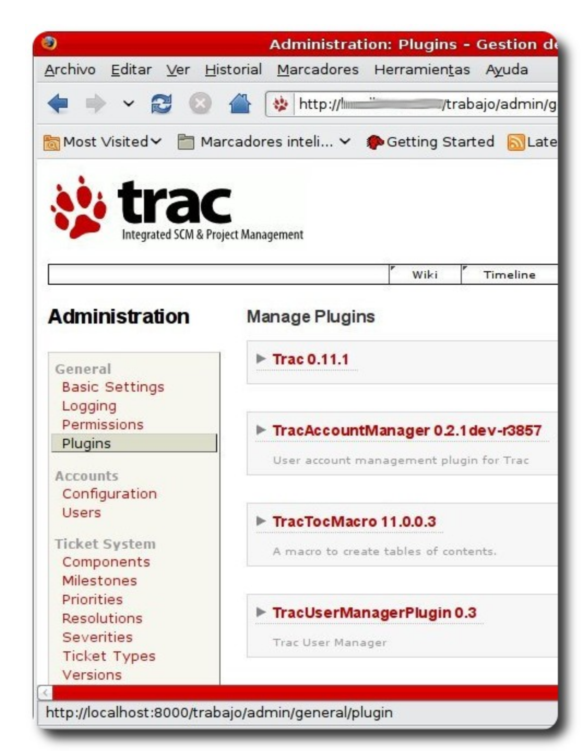


Reset de la contraseña

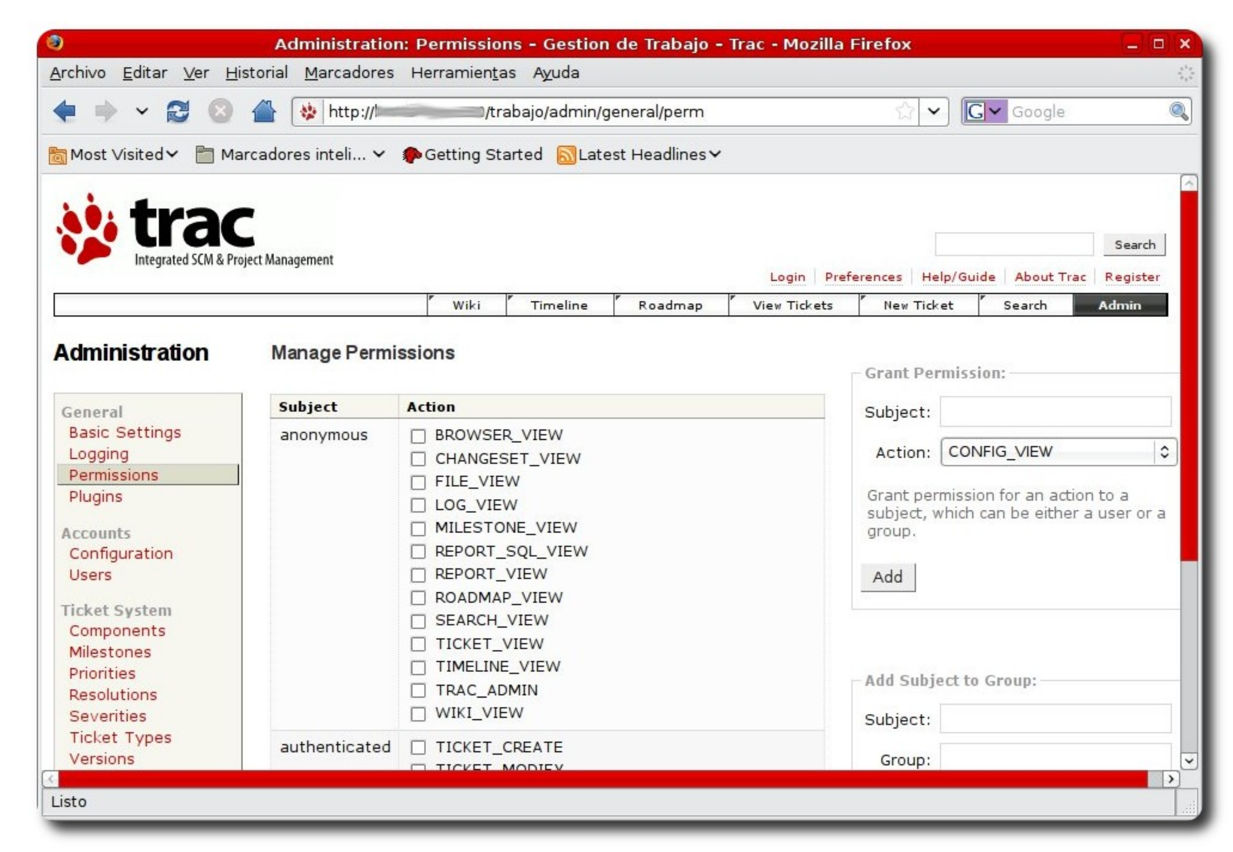




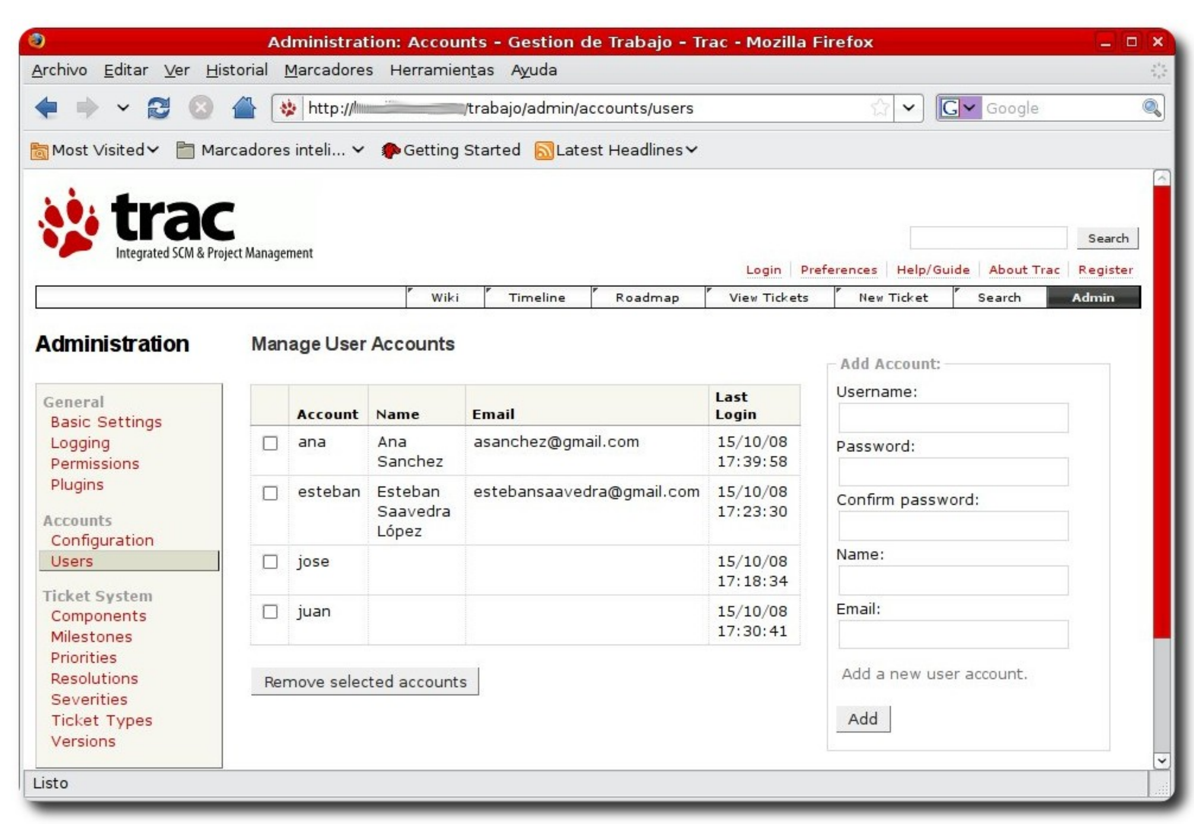
Personalizar información del proyecto



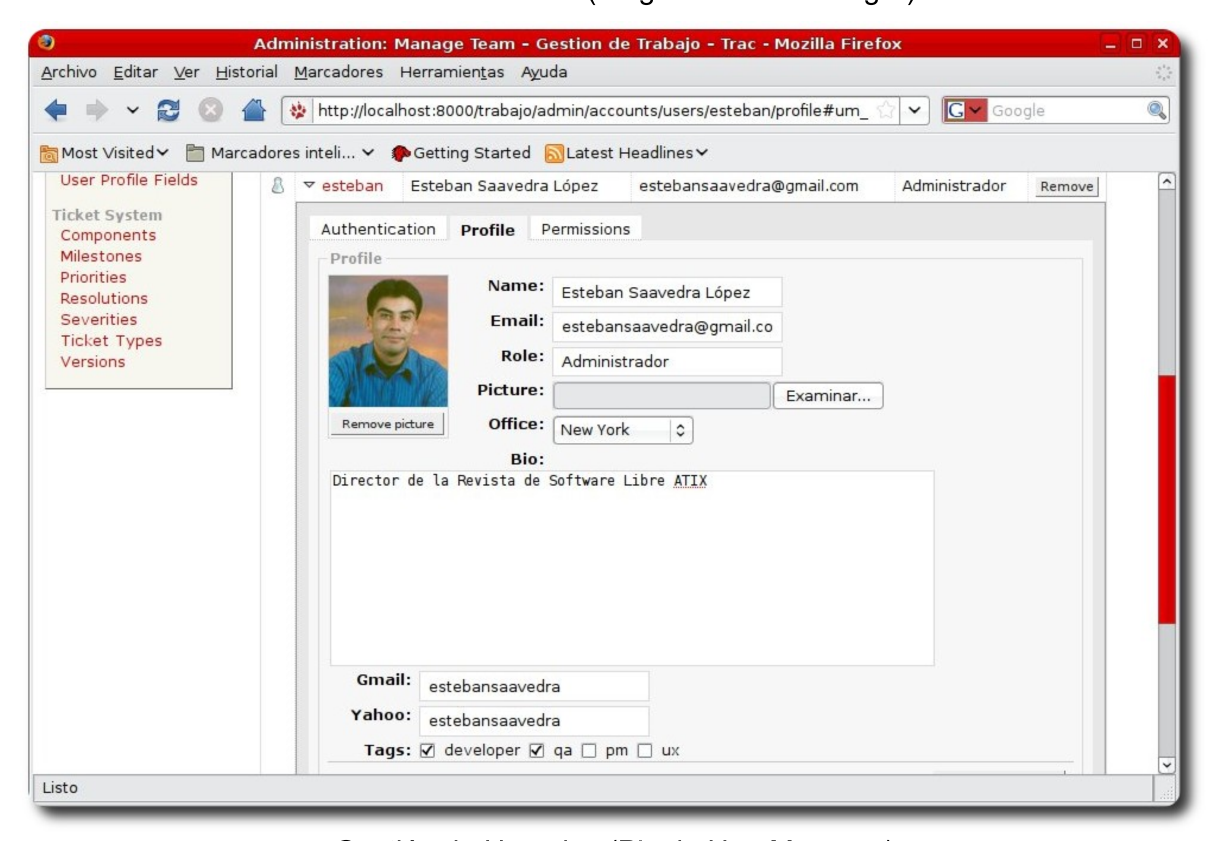
Gestión de Plugins



Gestión de Permisos



Gestión de Usuarios (Plugin AccountManager)



Gestión de Usuarios (Plugin UserManager)

Interacción y flexibilidad

Para concluir esta entrega podemos mencionar la enorme flexibilidad que tiene la gestión de **Trac**, ya sea desde línea de comandos o desde su interfaz Web. Flexibilidad que nos permite adquirir más y mayores funcionalidades de las provistas inicialmente, entre las que podemos citar:

- ✔ Interacción con distintos sistemas de manejo de versiones (bazaar, git, mercurial)
- ✔ Manejo de eventos y comunicación por medio de email
- ✔ Uso de themes para cambiar la apariencia
- ✔ y mucho más

Recuerden que **Trac** por defecto viene con una guía que nos enseña y demuestra las posibles variantes a utilizar, ésta puede ser accedida por medio de la wiki del proyecto que se gestiona.

Por último invitarlos a probar e investigar más detalles de las capacidades de esta herramienta, que se ha convertido realmente en un valioso aporte al momento de gestionar un proyecto de desarrollo de software.

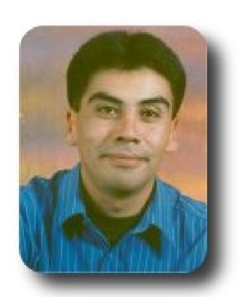
Referencias

[1] Proyecto **Trac**: http://trac.edgewall.org/

[2] Componentes adicionales: http://trac-hacks.org/

[3] Acceso a repositorios libres: https://opensvn.csie.org/

Autor



Esteban Saavedra López

Líder de la Comunidad ATIX (Oruro – Bolivia)
Activista de Software Libre en Bolivia
jesaavedra@opentelematics.org
http://jesaavedra.opentelematics.org


ATIX
Noticias

8vo Congreso Nacional de Software Libre Bolivia



Los días 13,14 y 15 la comunidad de Software Libre en Bolivia se vistió de gala, ya que celebró su **8vo Congreso Nacional de Software Libre**, realizado en la ciudad de La Paz.

Una vez mas, este evento acogió a profesionales, estudiantes, entusiastas y publico en general interesados en el Software Libre.



El afiche principal

8vo Congreso Nacional de Software Libre Bolivia

La **Revista Atix**, realizó la cobertura de este grandioso evento, en un futuro muy próximo tenemos la idea de sacar un número especial que describa y detalle todas y cada una de las actividades realizadas en este congreso.



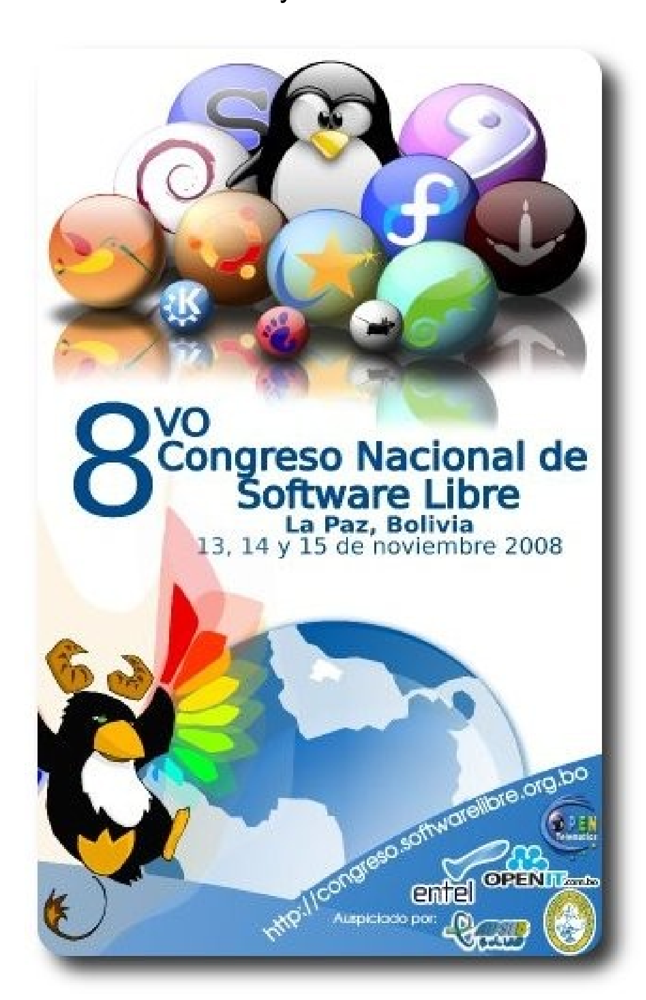
Gigantografía de la Revista Atix y el proyecto Opentelematics



Gigantografía del evento

8vo Congreso Nacional de Software Libre Bolivia

En esta oportunidad también se aprovecho de realizarse de forma paralela el encuentro de algunas comunidades especificas, como es el caso de la comunidad de **BoliviaOS**, comunidad de desarrolladores y colaboradores de la distribución de linux con sabor boliviano.



Gigantografía del evento



Gigantografía de BoliviaOS

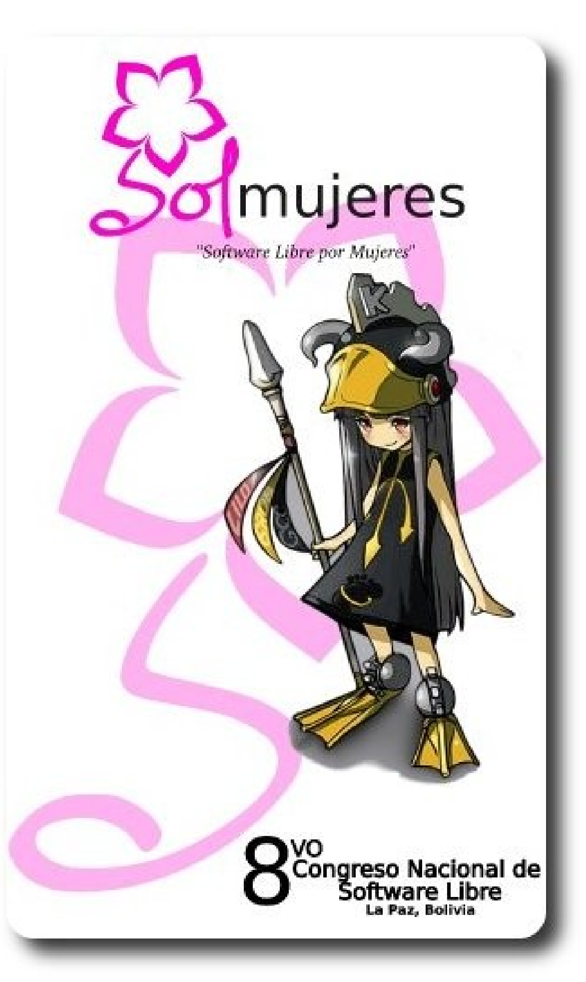
8vo Congreso Nacional de Software Libre Bolivia

La comunidad de **Ubuntu Bolivia**, también se hizo presente en este evento para mostrar las actividades y adelantos que poseen.

Por primera vez se realizo el encuentro de la comunidad **SolMujeres**, comunidad que vio la luz este 2008.



Gigantografía de Ubuntu Bolivia



Gigantografía de SolMujeres

8vo Congreso Nacional de Software Libre Bolivia

En esta ocasión el evento contó con el auspicio de importantes empresas e instituciones como Entel, Universidad Católica de Bolivia (Carrera de Ingeniería de Sistemas), Opentelematics Internacional en Bolivia, ADSIB – Agencia para el desarrollo de la Sociedad de la Información en Bolivia, OpenIT; todas ellas brindaron su apoyo y respaldo al evento mas importante del software Libre en Bolivia.



Auspiciadores del 8vo Congreso Nacional de Software Libre

InfoNews – Doble Clic

Las nuevas versiones de los navegadores siguen luchando, nadie se queda atrás



Mozilla, Opera, Google Chrome han liberado nuevas versiones muchas de ellas aún en su fase beta, tal es el caso de Mozilla que recientemente sacó la versión 3.1 beta 2 con el que se puede apreciar una nueva interfaz, una notable mejora en cuanto a la velocidad de carga de las páginas, un corrector ortográfico, marcadores, un bloqueador de ventanas emergentes, un gestor de descarga, además se tiene la característica de usar la función drag and drop lo que te permite arrastrar las pestañas y de este modo se pueda cambiar la posición de una ventana a otra y por ejemplo si se tiene una sola ventana, se puede arrastrar una pestaña y soltarla en el escritorio y se te abrirá una ventana nueva, parece algo sencillo pero en realidad estos detalles son importantes para los usuarios, otro aspecto importante es que ocupa menos memoria. Pruébalo tú mismo descargando la versión de Mozilla 3.1 beta2.



Otro de los navegadores que permanecen en la lucha es Opera que ahora lanzó la versión 10 alpha y lo que caracteriza a esta nueva versión es su velocidad de carga y el hecho de haber pasado la prueba Acid3 obteniendo un alto puntaje eso significa que esta versión puede soportar lo último en diseño web te hablo de Javascript, Document Objet Model y CCS (Cascading Style Sheets) entre otras.



Google Chrome acaba de lanzar su version definitiva, con la que nos trae nuevas características en cuanto a velocidad, mejoras en el bloqueador de Popups, mejoras en la administración de favoritos y por cierto esta versión está en español; aún no tenemos la versión final de Google Chrome que parece ser prometedor. ¿Chrome logrará colmar las expectativas de los usuarios?

A un paso de Fedora 11



Y seguimos con nuevas versiones ahora es el turno de Fedora que hace una par de semanas lanzó la versión 10 con nuevas características por ejemplo presenta un arranque más veloz, instalaciones virtuales remotas, además presenta el kernel 2.6.27, RPM 4.6, KDE 4.1, GNOME 2.24 y otras características innovadoras en esta versión, apenas se había liberado la versión 10 ahora se habla de Fedora 11, en la página **http://fedoraproject.org/** podrás encontrar las nuevas funcionalidades de la futura versión y además de un cronograma de desarrollo de Software con fechas tentativas y esperemos que estas no sufran ningún cambio.



Autor



Ivonne Karina Menacho Mollo
Titulada de la carrera de Ingeniería de Sistemas e Informática (F.N.I.)
Conductora programa radial “**Doble Clic**”
ivonnekarina2003@hotmail.com



InfoNews – Doble Clic

Blockbuster se apoya en MediaPoint y Linux

La famosa cadena de alquiler de películas Blockbuster se apoya en MediaPoint y Linux para resurgir.



Después de haber sufrido una terrible caída en su negocio, BLOCKBUSTER una de las cadenas más competidoras en el servicio de alquiler de películas, renueva su misión de" *ayudar a la gente a transformar noches ordinarias en noches con Gran éxito en taquilla*" al combinar el servicio ON DEMAND con un dispositivo basado en Linux.

BLOCKBUSTER tras haber acusado a la piratería de ser la principal causa de su fracaso con el gran dicho "*si no puedes con ellos únetelos*" inauguró la sección Download, llamado Blockbuster On Demand, un servicio que permite a sus clientes descargar películas desde Internet con un cierto costo, para esta descarga se creó un reproductor-receptor es decir un Set-Top Box de medios digitales el cual servirá como puente entre la gran red y los televisores.

El nombre de este dispositivo es MediaPoint fabricado por 2Wire, es un centro multimedia con la capacidad de descargar películas a través de la red, también se puede aprovechar la red doméstica para visualizar vídeos y escuchar música almacenados en cualquier equipo de la red, gracias al soporte de UPnP (Universal Plug and Play) y DLNA (Alianza de Redes Digitales Vivas), además dispone de una salida HDMI (Interfaz Multimedia de Alta Definición), y si de conectividad se trata, ésta ofrece dos alternativas una Inalámbrica y otra por cable Ethernet, también cuenta con un navegador, permitiendo a sus usuarios navegar por la red desde su sofá.

Al igual que una computadora, MediaPoint también necesita de un sistema operativo para su funcionamiento. La diferencia básica sería que, este dispositivo necesita un sistema operativo en tiempo real debido a que operaciones como la decodificación de MPEG necesita que se realice al instante y adivinen que… todo este funcionamiento estará a cargo del sistema operativo LINUX!!!.

Sin duda alguna Blockbuster regresó con la idea de tomar las riendas y ser líder mundial en su rubro, pero esta vez con un nuevo concepto y apoyándose en muy buenas tecnologías como MediaPoint y Linux.



La forzosa migración a Red Flag causa polémica en China



En la ciudad Nanchang al sur de China están imponiendo el uso de Red Flag Linux en los CiberCafés, la razón que dan las autoridades chinas para esta obligatoria migración es “reducir el uso de software pirata como Windows”. Pero varios dueños de CiberCafés argumentaron contar con una licencia oficial del sistema operativo Windows, pero aun así se les obligó a sustituir este por Red Flag – Linux Chino.

Lo extraño es que los dueños de estos CiberCafés para instalar esta distribución de Linux deben pagar una licencia de 5000 yuan (US$720), siendo Red Flag una distribución supuestamente libre.



Nanchang no es la única ciudad que está sufriendo esta migración obligatoria, Fuzhou también fue afectada, ni tampoco esta es la única medida ordenada a los CiberCafés, la instalación de cámaras y escáneres para la identificación de usuarios en estos, fue orden gubernamental. Sin duda alguna, los dueños de estos CiberCafés están siendo muy afectados en el bolsillo.

Autor



Marcia Estrella Velasquez Aguilar
Egresada de la carrera de Ingeniería de Sistemas (F.N.I.)
mevaguerreradelaluz@gmail.com

InfoNews – Doble Clic

Songbird más que un reproductor multimedia



Songbird es un reproductor multimedia, basado en la plataforma XULRunner de Mozilla, por lo cual dispone de versiones para Windows, Mac OS X, y Linux.

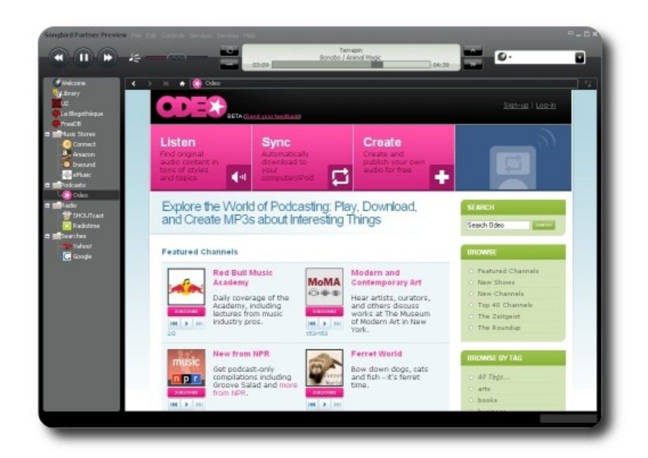
Songbird es un reproductor multimedia integrado a muchísimas cosas más, ya que trata de integrar las características de Firefox, Last.fm y iTunes

Al hacer uso de Songbird solo tienes que pensar en una canción, escribirla en un buscador que lleva songbird, y automáticamente te saldrá una web en la que puedes escuchar la canción entera de manera rápida y gratuita, con enlaces a sus letras, vídeos, fotos… pero con la facilidad para que la puedas descargar automáticamente a muy buenas velocidades (entre 200 y 300kb/s).

Según Mozilla, songbird está “*destinado a reproducir la música que quieras, de los sitios que quieras, en los dispositivos que quieras, desafiando las convenciones del descubrimiento, compra, consumo y organización de la música en Internet*”

Entre sus características principales se encuentra:

- ✔ GStreamer: un framework multimedia libre multiplataforma
- ✔ mashTape: Al escuchar música con Songbird podrás ver fotos de Flickr, vídeos de Youtube, biografías en last.fm, noticias en Google y muchas otras cosas relacionadas con el artista de la canción (integración con Last.fm).
- ✔ Listas de reproducción inteligentes reutilizables.
- ✔ Entradas para conciertos: Puedes ver los próximos eventos en tu localidad, basados en los artistas de tu librería de música.
- ✔ Capacidad de reproducir archivos de múltiples formatos, como MP3, AAC, Ogg Vorbis, FLAC y Windows Media Audio.
- ✔ Interfaz personalizable. Una interfaz gráfica de usuario configurable y plegable, así como el modo de mini-reproductor.
- ✔ La capacidad de suscribirse a blogs de mp3 así como listas de reproducción.
- ✔ Favoritos creados por el usuarios.
- ✔ Capacidad de construir mezclas personalizadas.
- ✔ Add-ons a la carta.



AWS acrecienta su oferta de servicios web en linea



Amazon AWS sigue creciendo en la oferta de servicios web, después del éxito en sus anteriores servicios Web basados en “**computación en la nube**” que los usuarios y las compañías pueden usar para construir aplicaciones, servicios como:

- ✔ Elastic Compute Cloud (EC2)
- ✔ Simple Storage Service (Amazon S3)
- ✔ Simple Queue Service (Amazon SQS)
- ✔ Flexible Payments Service (Amazon FPS)
- ✔ y el recientemente lanzado CloudFront.

Amazon SimpleDB™ BETA

Amazon aws presenta a **SimpleDB**, que es un servicio que permite ejecutar consultas sobre datos estructurados en tiempo real, sin la complejidad de las bases de datos relacionales, brindando una opción, para quienes necesiten una base de datos a medida, fácil de usar y sin un elevado costo de mantenimiento.

Amazon **SimpleDB**, un servicio Web que provee funciones medulares de base de datos tales como indexación y consulta.

Algunas características de su lanzamiento, será gratis por un período de 90 en su versión beta, se podrá consumir más de 500 MB de almacenamiento y puede usar más de 25 horas máquina cada mes, transferencia de 1 GB de datos de entrada y otro GB de salida. Una vez concluido el tiempo de prueba, este servicio se estima tendrá un costo reducido y asequible a la mayoría de los usuarios.

Aunque Amazon AWS fue un pionero en el mercado de la infraestructura de computación en la nube, muy pronto enfrentara la competencia, el más próximo es Microsoft con su servicio Windows Azure

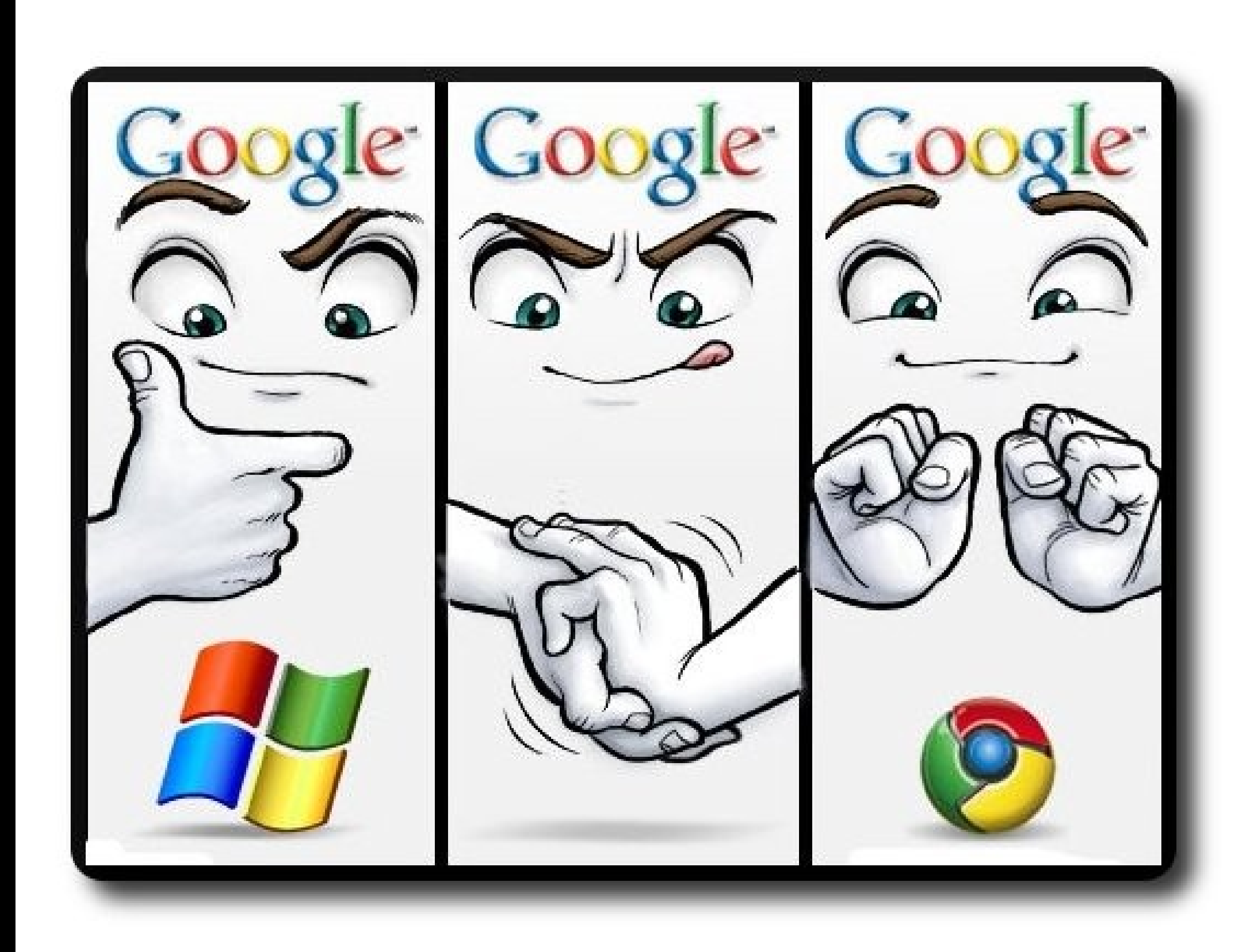


Autor

Jenny Saavedra López
Diseño y Edición Revista Atix
jennysaavedra@gmail.com



Comics





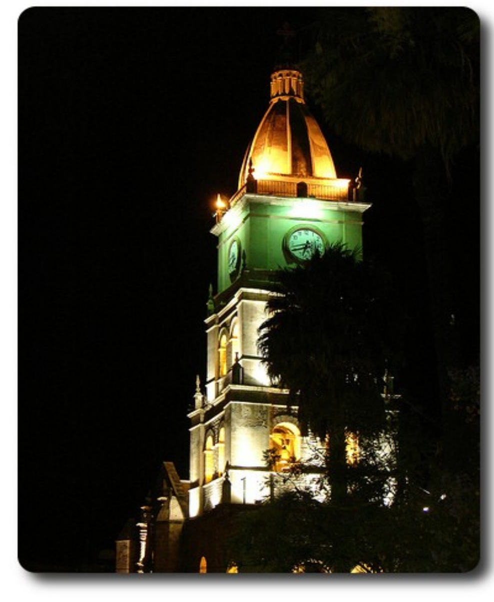
Conociendo lo Nuestro

Cochabamba

Turismo y Libertad



Cristo de La Concordia



Catedral de Cochabamba



Vista nocturna de la ciudad

Turismo y Libertad



Centros recreacionales de Cochabamba



Panorámica de la Ciudad



Teleférico del cristo de la concordia

Libres para pensar, libres para decidir, libres para crear



Te ofrecemos este espacio para mostrar tu Creatividad

ATIX

La Fuerza del Software Libre

Envíanos tus diseños y creaciones para publicarlos



Desea que en estas fiestas de fín de año
las bendiciones del niño Dios,
lleguen a tu hogar
traendo consigo mucha
Felicidad y los mejores deseos
de prosperidad para el
año que viene.



Consultas

Contacto

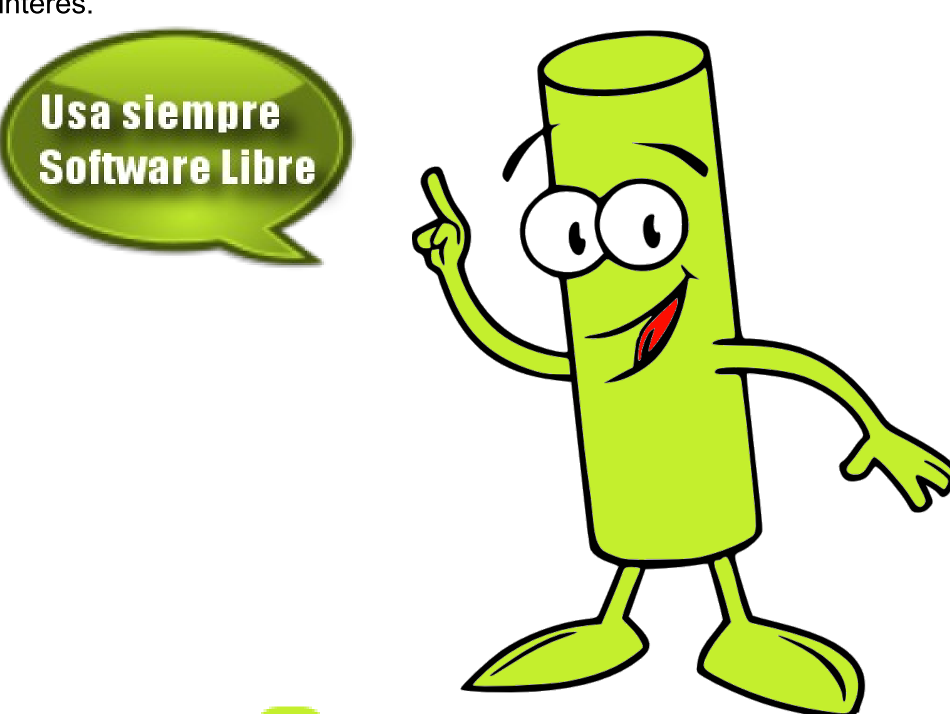
Para solicitar cualquier información, puedes contactar a:

- Esteban Saavedra López (jesaavedra@opentelematics.org)
- Williams Chorolque Choque (williamsis@gmail.com)

Publicación

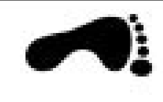
Te invitamos a ser parte de la **Revista ATIX**. La forma de participar puede ser enviándonos:

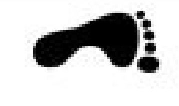
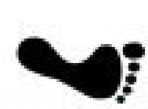
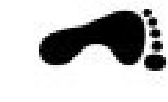
- Artículos referidos a áreas como:
  - Instalación y personalización de Aplicaciones
  - Scripting
  - Diseño gráfico
  - Programación y desarrollo de aplicaciones
  - Administración de servidores
  - Seguridad
  - y cualquier tema enmarcado dentro del uso de Software Libre
- Trucos y recetas.
- Noticias.
- Comics.
- Links de interés.





Marcamos Huella




http://atix.opentelematics.org